ESCUELA POLITÉCNICA
SUPERIOR DE CÓRDOBA
Universidad de Córdoba

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE RECONOCIMIENTO FACIAL PARA CONTROL DE ACCESO

Autor: Francisco Javier Ortiz Herrerías
Directores: D. Joaquín Olivares Bueno

septiembre, 2025

UNIVERSIDAD
D
CÓRDOBA

Agradecimientos

Agradezco a mi tutor, Joaquín Olivares Bueno, por su orientación en las fases necesarias para la realización de este Trabajo Fin de Grado.

Quiero dar también las gracias a mi familia, por su apoyo constante y por darme el ánimo que he necesitado para llegar hasta aquí.

Finalmente, a mis amigos, por estar a mi lado en este camino y recordarme la importancia de seguir adelante con esfuerzo y motivación.

Resumen

Este trabajo de fin de grado propone la creación e implementación de un prototipo de sistema de control de acceso mediante reconocimiento facial, basado en el uso de un ESP32-CAM y un servidor backend. El sistema se diseñó con dos modos de procesamiento: uno local, donde el ESP32-CAM realiza el reconocimiento facial directamente en el dispositivo, y otro remoto, donde el dispositivo envía la imagen a un servidor FastAPI que gestiona la detección y reconocimiento. Uno de los pilares fundamentales de este trabajo es la comparación de ambos modos, evaluada mediante distintas métricas con el fin de determinar cuál resulta más adecuado en cada situación.

Además, se han desarrollado dos interfaces de usuario distintas: una en la que todo se centraliza en una aplicación desarrollada con Flutter, y otra en la que se hace uso de un entorno web cargado en el propio dispositivo, reservando la aplicación únicamente para la consulta de registros. Estos registros se almacenan en una base de datos SQLite, mientras que la información de los rostros se guarda como vectores de datos en archivos binarios optimizados que permiten una administración más eficaz y portable.

En conjunto, este prototipo constituye una solución accesible y eficaz para la gestión de accesos mediante reconocimiento facial, garantizando seguridad, eficiencia y trazabilidad en entornos de pequeña y mediana escala, y sentando las bases para futuras mejoras orientadas a escenarios reales.

Palabras clave: *Reconocimiento facial, Control de acceso, ESP32-CAM, FastAPI, Flutter.*

Abstract

This TFG proposes the design and implementation of a prototype facial recognition access control system, based on the use of an ESP32-CAM and a backend server. The system was designed with two processing modes: a local mode, where the ESP32-CAM performs facial recognition directly on the device, and a remote mode, where the device sends the captured image to a FastAPI server that handles detection and recognition. One of the main pillars of this work is the comparison between both modes, evaluated through different metrics in order to determine which approach is more suitable in each situation.

In addition, two different user interfaces were developed: one centralized in a Flutter application, and another based on a web environment served directly by the device, with the mobile application being used only for record visualization. Access records are stored in a SQLite database, while the facial data is saved as optimized binary vectors, allowing for more efficient and portable management.

Overall, this prototype provides an accessible and efficient solution for access management through facial recognition, ensuring security, efficiency, and traceability in small and medium-scale environments, while laying the foundation for future improvements aimed at real-world scenarios.

Keywords: *Facial recognition, Access control, ESP32-CAM, FastAPI, Flutter.*

Índice general

Índice de Figuras

Índice de Tablas

Acrónimos

API	Application Programming Interface (Interfaz de Programación de Aplicaciones).
BLE	Bluetooth Low Energy.
CNN	Convolutional Neural Network (Red Neuronal Convolucional).
DB	Database (Base de Datos).
EFLD	Efficient Facial Landmark Detection.
ESP-WHO	Framework de reconocimiento facial de Espressif.
ESP32	Microcontrolador de 32 bits de Espressif.
ESP32-CAM	Variante del ESP32 con cámara integrada.
FastAPI	Framework web en Python para APIs rápidas.
FHE	Fully Homomorphic Encryption (Cifrado Homomórfico Completo).
GPIO	General Purpose Input/Output.
HTTP	HyperText Transfer Protocol.
IDE	Integrated Development Environment.
IoT	Internet of Things (Internet de las Cosas).
IR-Shuffle	Inverted Residual Shuffle Unit.
JSON	JavaScript Object Notation.
LCNN	Lightweight Convolutional Neural Network (Red Convolucional Ligera).
LED	Light Emitting Diode.

LFW	Labeled Faces in the Wild (conjunto de datos de rostros en entornos no controlados).
MTCNN	Multi-task Cascaded Convolutional Neural Network.
OMTCNN	Optimized Multi-task Cascaded Convolutional Neural Network.
OTA	Over-The-Air (actualización remota).
PAIR	Low-power, Efficient and Accurate Facial-Landmark Detection for Embedded Systems Challenge (competición del IEEE ICME 2024).
PSRAM	Pseudo Static RAM.
RAM	Random Access Memory.
SDK	Software Development Kit.
SQLite	Motor de base de datos ligero.
SRAM	Static Random Access Memory.
TFG	Trabajo de Fin de Grado.
UI	User Interface (Interfaz de Usuario).
UX	User Experience (Experiencia de Usuario).
VGA	Video Graphics Array.
Wi-Fi	Wireless Fidelity.
WPA2	Wi-Fi Protected Access 2.

Capítulo 1

Introducción

1.1. Contexto

El control de acceso es un componente fundamental en cualquier lugar donde la seguridad sea prioritaria, desde instalaciones industriales hasta comunidades residenciales. Su finalidad es garantizar que únicamente las personas autorizadas entren en zonas restringidas.

Tradicionalmente esta función se ha resuelto con llaves, tarjetas magnéticas o incluso códigos numéricos. El problema de estos métodos es que, por ejemplo, una llave puede perderse o duplicarse, una tarjeta puede ser robada y un código transmitido sin control.

Ante estas limitaciones, han surgido alternativas más modernas que involucran la biometría. Sistemas basados en huellas dactilares, iris o reconocimiento facial se han popularizado enormemente en los últimos años. Entre ellas, el reconocimiento facial destaca por ser muy rápido y cómodo para el usuario, además de no requerir contacto físico. Con los últimos avances en visión por computador, se ha abaratado el hardware necesario, estando al alcance de proyectos de todo tipo.

Sin embargo, los sistemas comerciales que incorporan esta técnica suelen tener precios elevados, dependientes de licencias propietarias, lo que limita su adopción en pequeñas empresas o en proyectos académicos.

1.2. Alcance

Este trabajo se enmarca en la búsqueda de soluciones más accesibles y plantea el desarrollo de un sistema de control de acceso por reconocimiento facial que sea económico, escalable y fácil de desplegar.

La propuesta se apoya en el uso del Variante del ESP32 con cámara integrada (ESP32-CAM), un microcontrolador con cámara integrada y conectividad Wireless Fidelity (Wi-Fi), y un servidor backend desarrollado en Framework web en Python para APIs rápidas (FastAPI). Con esta arquitectura se plantean dos enfoques complementarios:

- **Modo local:** el propio dispositivo realiza la detección y el reconocimiento facial.
- **Modo remoto:** el dispositivo captura la imagen y delega el procesamiento al servidor, que se encarga de aplicar modelos más avanzados.

El sistema se completa con una aplicación móvil desarrollada en Flutter y una interfaz web ligera alojada en el ESP32-CAM, que permite gestionar usuarios y consultar registros de accesos de manera sencilla. De este modo, se obtiene una solución que abarca todo el flujo, desde la captura de imágenes hasta la administración y visualización de resultados.

No solo se busca una viabilidad técnica, sino también responder a una demanda creciente de soluciones de seguridad sin contacto, acentuada por la digitalización y la necesidad de minimizar la interacción física en entornos cotidianos.

En definitiva, este proyecto pretende demostrar que es posible construir un sistema de control de acceso facial funcional y fiable utilizando hardware de bajo coste y herramientas de software abiertas. Además, se aporta un análisis comparativo de los dos modos de operación, enriqueciendo el trabajo desde una perspectiva más académica. Aunque se plantea como un prototipo, su diseño modular sienta las bases para su ampliación e integración en escenarios reales.

1.3. Objetivos

El objetivo general de este trabajo es diseñar e implementar un sistema de control de acceso basado en reconocimiento facial que combine bajo coste, escalabilidad y fiabilidad, e integrar en él dos modos de procesamiento diferenciados para analizar sus ventajas y limitaciones.

Objetivos específicos

1. Integrar un módulo ESP32-CAM para captura de imágenes en tiempo real, priorizando un funcionamiento estable.
2. Implementar el reconocimiento facial en dos modos diferenciados:
   - **Local:** procesamiento únicamente en el ESP32-CAM.
   - **Remoto:** procesamiento en el servidor.
3. Desarrollar un servidor backend encargado de la gestión de los usuarios y de los registros, además del procesamiento en el modo remoto.
4. Diseñar una interfaz web para la configuración de los modos.
5. Diseñar e implementar una aplicación móvil como interfaz principal para la visualización de registros y administración.
6. Evaluar el rendimiento del sistema, midiendo la precisión, velocidad, capacidad, escalabilidad, entre otras métricas en ambos modos.

Capítulo 2

Antecedentes

2.1. Proyectos de referencia

Antes de empezar con el desarrollo es importante entender cómo está el panorama actual, investigando tecnologías y metodologías que se han usado para resolver problemas similares.

El más importante para el entendimiento de las capacidades del ESP32-CAM es "ESP32-CAM Access Control System". Este es un proyecto disponible en GitHub que implementa un sistema de control de acceso con detección facial en ESP32-CAM programado en Arduino Integrated Development Environment (IDE). El dispositivo era capaz de reconocer rostros y accionar un mecanismo físico de apertura. Este proyecto resultó especialmente útil para entender las capacidades reales del Microcontrolador de 32 bits de Espressif (ESP32) ya que permitió comprobar que la librería de reconocimiento facial que finalmente se utilizó en este Trabajo de Fin de Grado (TFG) requiere una versión más antigua de las bibliotecas de hardware Arduino ESP32, ya que las versiones más recientes no soportan el reconocimiento facial. A partir de esta base, el presente TFG evolucionó a un sistema mucho más robusto incorporando un backend para el reconocimiento de forma remota, una base de datos para gestionar usuarios y registros, y por último una aplicación móvil para el control remoto [1].

Otro ejemplo clave fue "Camera Web Server", incluido como referencia en el Software Development Kit (SDK) de Espressif. Este proyecto implementa la captura de imágenes desde la cámara integrada y las transmite mediante una interfaz web básica. También implementa una pequeña prueba de detección y reconocimiento facial. Aunque sencillo, este ejemplo fue clave en los comienzos para entender la

inicialización de la cámara y transmisión de imágenes [2].

Estos proyectos confirmaron que el ESP32-CAM sirve como base válida para un sistema de reconocimiento facial, aunque dejaron claro que sus recursos son limitados.

2.2. Revisión científica de artículos recientes

La investigación científica en los últimos años ha centrado sus esfuerzos en diseñar modelos ligeros y eficientes para el reconocimiento facial en sistemas embebidos, con el objetivo de superar las restricciones de memoria, potencia de cálculo y consumo energético.

Lv et al. (2021) proponen un sistema multicore embebido que combina un detector facial optimizado (Optimized Multi-task Cascaded Convolutional Neural Network (OMTCNN)) con una Lightweight Convolutional Neural Network (Red Convolucional Ligera) (LCNN) ligera, alcanzando un 98,13 % de precisión en Labeled Faces in the Wild (conjunto de datos de rostros en entornos no controlados) (LFW) y multiplicando por 6,4 la velocidad de procesamiento respecto a modelos más pesados. Este tipo de optimizaciones refuerza la validez de los enfoques basados en Multi-task Cascaded Convolutional Neural Network (MTCNN), como los empleados en la librería Framework de reconocimiento facial de Espressif (ESP-WHO), utilizada en este TFG para el reconocimiento local en el dispositivo ESP32-CAM [3].

En esta misma línea, Chen et al. (2023) presentan un modelo ligero basado en bloques Inverted Residual Shuffle Unit (IR-Shuffle), que con apenas 1,45 MB logra un 98,65 % de precisión en LFW y supera a MobileFaceNet en tiempo de inferencia. Implementado en un Jetson Nano, valida la posibilidad de desplegar sistemas precisos y en tiempo real en hardware de bajo coste [4].

Otros trabajos abordan tareas específicas del pipeline de reconocimiento. Wu (2024) presenta el modelo Efficient Facial Landmark Detection (EFLD), ganador del reto IEEE ICME 2024 Low-power, Efficient and Accurate Facial-Landmark Detection for Embedded Systems Challenge (competición del IEEE ICME 2024) (PAIR) Challenge, diseñado para detección de landmarks en entornos con recursos limitados, combinando bajo consumo, robustez y entrenamiento multiformato para mejorar la generalización [5].

Desde la perspectiva de optimización en entornos reales, Andriyanov y Dementiev (2024) muestran que detectores ligeros como YuNet, combinados con funciones de pérdida avanzadas como ArcFace loss, permiten mejorar precisión y velocidad en

edge computing, lo que resulta crítico en dispositivos como videoporteros inteligentes o sistemas de videovigilancia [6].

En cuanto a aplicaciones prácticas, Abderraouf et al. (2024) desarrollan un sistema de monitorización de asistencia en aulas basado en Raspberry Pi, evidenciando la viabilidad de integrar reconocimiento facial en entornos educativos reales pese a las limitaciones del hardware [7]. De forma complementaria, Nguyen-Tat et al. (2024) implementan un sistema de gestión de asistencia en Jetson Nano empleando el clásico Haar Cascade + OpenCV2, lo que muestra que algoritmos tradicionales siguen siendo útiles en escenarios embebidos donde prima la eficiencia sobre la precisión extrema [8].

Finalmente, en el ámbito de la seguridad y privacidad, Serengil y Ozpinar (2025) introducen CipherFace, un marco de cifrado homomórfico (Fully Homomorphic Encryption (Cifrado Homomórfico Completo) (FHE)) que permite realizar cálculos de similitud sobre embeddings faciales cifrados en la nube, garantizando la protección de datos biométricos sensibles durante el procesamiento [9].

En conjunto, esta revisión evidencia una tendencia clara: trasladar el reconocimiento facial desde servidores de alto rendimiento hacia dispositivos embebidos ligeros, manteniendo un equilibrio entre precisión, velocidad y consumo. Al mismo tiempo, plantea nuevos retos en materia de seguridad, anti-spoofing y privacidad, aspectos que se abordan en este TFG mediante un enfoque híbrido que combina el ESP32-CAM con un servidor remoto para compensar sus limitaciones.

2.3. Sistemas comerciales

Además de los proyectos abiertos, el mercado dispone de soluciones comerciales consolidadas que marcan un estándar en términos de fiabilidad y prestaciones. Entre las empresas especializadas más consolidadas a nivel internacional destacan ZKTeco, Suprema o Hikvision, reconocidas por la fabricación de terminales avanzados que integran reconocimiento facial entre otras funciones de seguridad complementarias. Estos fabricantes se caracterizan por ofrecer dispositivos robustos, con alta capacidad de procesamiento y orientados a entornos corporativos o industriales.

En concreto, los modelos más representativos y parecidos a nuestro proyecto resultante son los siguientes:

- **SpeedFace-V5L (ZKTeco):** este terminal de acceso biométrico ofrece un reconocimiento ultrarrápido con luz visible e integra algoritmos anti-spoofing frente a rostros, videos o máscaras. Además, dispone de gran capacidad de

almacenamiento para usuarios, detección de temperatura corporal y proporciona una fácil integración en cerraduras electrónicas. Su precio oscila entre 300 y 500 euros dependiendo de su proveedor [10, 11].

- **Suprema FaceLite FL-DB (Suprema):** dispositivo compacto que combina rapidez y precisión en la identificación, incluso en condiciones extremas de iluminación, pudiendo funcionar tanto en completa oscuridad como en máxima luminosidad. Ofrece soporte para hasta 30 000 usuarios y mantiene tiempos de respuesta muy breves. Su coste en mercado se sitúa entre 800 y 1000 euros [12].

Estos sistemas comerciales sirven como referencia para evaluar la viabilidad de un sistema de menor coste y tamaño reducido, como el que se propone en este proyecto. Presentan gran fiabilidad y prestaciones avanzadas pero a un precio muy elevado, con dependencia de software propietario y escasa flexibilidad para proyectos de menor escala. Estas barreras refuerzan la necesidad de explorar alternativas más asequibles y adaptables, como la solución propuesta en este trabajo.

2.4. Tecnologías hardware

En el desarrollo de un sistema de control de acceso por reconocimiento facial es fundamental elegir bien el hardware. De esa elección dependen factores como el coste, el consumo energético, la potencia de cálculo disponible y lo sencillo que resulte integrar cada opción en el diseño general.

Una de las primeras alternativas que se valoraron fue la Raspberry Pi. Se trata de un ordenador monoplaca muy popular en proyectos educativos y de Internet of Things (Internet de las Cosas) (IoT). Su principal ventaja es la potencia: puede ejecutar modelos completos de inteligencia artificial en local, lo que la convierte en una plataforma muy flexible. Ahora bien, no es perfecta. Su precio suele estar entre los 40 y los 80 euros, consume más energía que otras opciones y, además, su tamaño es mayor, lo que resta portabilidad. Para un prototipo barato y compacto no encajaba del todo [13].

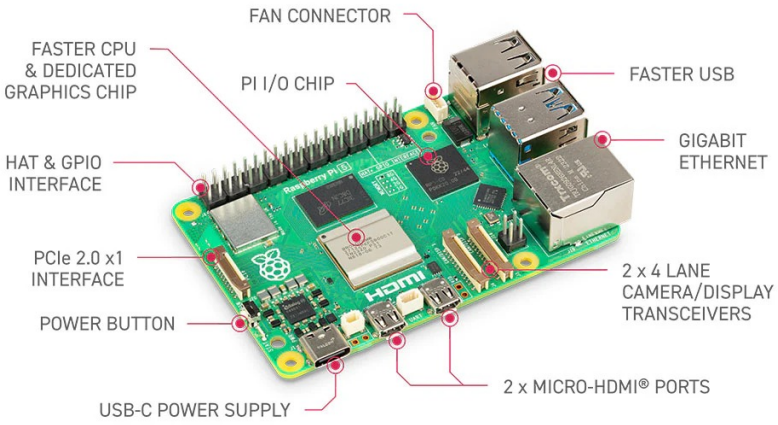

Figura 2.1: Placa Raspberry Pi con identificación de sus principales puertos y componentes.

El segundo candidato fue el ESP32, un microcontrolador de Espressif que ya integra Wi-Fi y Bluetooth Low Energy (BLE) en el propio chip. Es pequeño, eficiente y muy económico (en torno a 10–15 euros). Para muchos proyectos IoT es la base ideal. El inconveniente es que no incluye cámara, de modo que sería necesario añadir periféricos externos, encareciendo y complicando la solución [14].

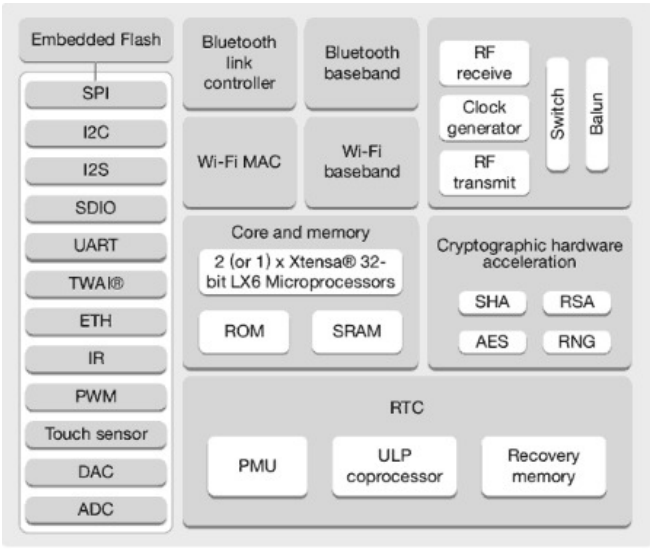

Figura 2.2: Diagrama de bloques de la arquitectura del ESP32, mostrando procesadores, memorias, periféricos y módulos de comunicación.

Finalmente se evaluó el ESP32-CAM, una variante del ESP32 que sí incluye cámara integrada y soporte para tarjetas microSD. Esta característica simplifica mucho el diseño, ya que en un solo módulo se combina el microcontrolador con el sensor de imagen. Cuesta entre 5 y 10 euros, consume poco y ocupa muy poco espacio. Como desventaja, tiene una memoria limitada y no puede ejecutar modelos de inteligencia artificial complejos directamente en local [15].

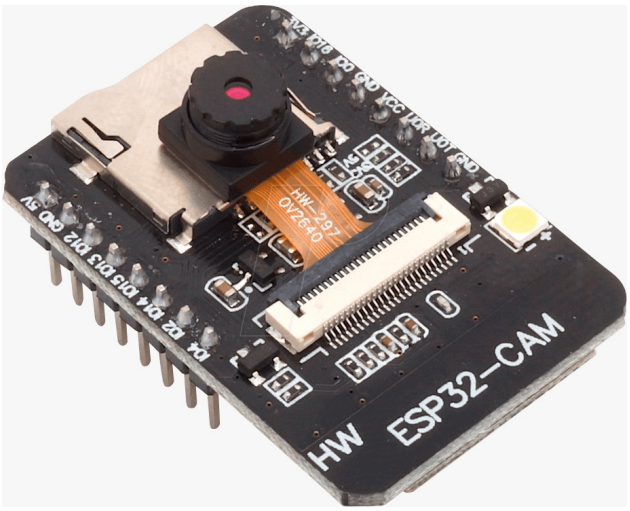

Figura 2.3: Módulo ESP32-CAM con cámara integrada y ranura para tarjeta microSD.

Pese a esas limitaciones, el ESP32-CAM fue el elegido para este proyecto. La razón es sencilla: ofrece la mejor relación entre precio, tamaño y prestaciones. Y aunque su potencia no permite trabajar con modelos avanzados, gracias a la conectividad Wi-Fi se puede complementar con un servidor remoto, lo que abre la puerta a un sistema híbrido que combina simplicidad en el dispositivo y potencia en la nube.

2.5. Tecnologías software y librerías

Además del hardware, el software juega un papel clave en un sistema de reconocimiento facial. La elección de librerías y frameworks determina no sólo la precisión del sistema, sino también su facilidad de desarrollo y su capacidad para escalar. En este proyecto se analizaron diferentes opciones, tanto para el procesamiento en el servidor como en el dispositivo local (ESP32-CAM).

Procesamiento en servidor

Una de las primeras alternativas estudiadas fue OpenCV, la biblioteca de visión por computador más conocida. Es de código abierto, multiplataforma y compatible con varios lenguajes, entre ellos Python y C++. Incluye algoritmos para detección de objetos, análisis de movimiento o reconstrucción 3D, lo que la convierte en una opción muy flexible. Además, cuenta con una comunidad enorme y abundante documentación. Su punto débil es que su curva de aprendizaje puede ser desafiante y es difícilmente optimizable en dispositivos con recursos limitados [16].

La otra opción fue InsightFace, un framework más moderno y especializado, diseñado para el reconocimiento facial con aprendizaje profundo. Ofrece modelos preentrenados con gran precisión y permite trabajar con volúmenes de usuarios mucho más altos que OpenCV. Como contrapartida, requiere más recursos de cálculo y su instalación es un poco más exigente. Aun así, para este trabajo se eligió InsightFace como la opción principal en el modo remoto, ya que compensa esas exigencias con resultados mucho más precisos y escalables [17, 18].

Procesamiento en el ESP32-CAM

El análisis también incluyó librerías que permitieran realizar reconocimiento directamente en el dispositivo. En este sentido, se valoró TensorFlow Lite, pensado para ejecutar modelos reducidos en dispositivos móviles y embebidos. Aunque es ligero en comparación con TensorFlow estándar, sigue necesitando más memoria y potencia de la que puede ofrecer un ESP32-CAM [19].

Por otro lado, está ESP-WHO, la librería oficial de Espressif para tareas de visión en dispositivos de la familia ESP32. Está optimizada para trabajar con el ESP32-CAM y, aunque no tiene la precisión de frameworks más pesados, ofrece buen rendimiento local, bajo consumo y una integración sencilla. Por esa razón fue la seleccionada para implementar el modo local en este proyecto [20].

Entornos de desarrollo

En cuanto al entorno de programación, se consideraron Arduino IDE y PlatformIO. Arduino IDE fue finalmente el elegido por su sencillez y porque ofrece compatibilidad total con ESP-WHO. PlatformIO es más potente para proyectos grandes, con mejor gestión de dependencias y control de versiones, pero también requiere una configuración más compleja [21].

Frameworks web y móviles

Para el backend se analizaron varias opciones en Python. Flask es ligero y fácil de aprender, pero limitado en aplicaciones con muchas peticiones [22]. Django, en el otro extremo, es muy robusto y completo, aunque demasiado pesado para un backend que solo necesita ofrecer Application Programming Interface (Interfaz de Programación de Aplicaciones)s (APIs) [23]. Entre ambas apareció FastAPI, que combina ligereza y alto rendimiento, además de integrarse muy bien con librerías de aprendizaje profundo. Por estas razones se eligió como servidor en este proyecto [24].

En el caso de las aplicaciones móviles, se evaluaron React Native y Flutter. La primera, basada en JavaScript, permite reutilizar código entre Android e iOS, pero algunas integraciones requieren ajustes manuales que aumentan la complejidad [25]. Flutter, en cambio, permite crear aplicaciones multiplataforma con un solo código base y destaca por su rapidez en el desarrollo y su amplio catálogo de widgets. Esa facilidad de diseño y la rapidez para visualizar cambios en tiempo real llevaron a seleccionarlo como framework principal para la aplicación móvil [26].

Capítulo 3

Restricciones

Como en cualquier proyecto, el desarrollo de este sistema no está exento de limitaciones que condicionan tanto su diseño como sus resultados. Algunas de estas restricciones provienen del propio hardware utilizado, mientras que otras responden a las condiciones en las que se ejecuta el trabajo, como el tiempo disponible o el alcance definido para un TFG.

La primera y más evidente limitación está en el ESP32-CAM. Se trata de un dispositivo económico y compacto, pero con recursos reducidos. Su memoria y potencia de cálculo son muy limitadas en comparación con otros sistemas embebidos más potentes, como una Raspberry Pi. Esto significa que no es posible ejecutar modelos complejos de reconocimiento facial directamente en el dispositivo y obliga a buscar alternativas más ligeras, como ESP-WHO, o a derivar el procesamiento a un servidor externo.

La calidad de la cámara integrada en el ESP32-CAM también marca un límite. Su resolución es adecuada para prototipos y pruebas en entornos controlados, pero no alcanza el nivel de precisión de cámaras profesionales, lo que puede afectar al reconocimiento en condiciones de baja luz o con rostros alejados.

Otro factor importante es la conectividad Wi-Fi. El modo remoto depende por completo de la red para transmitir imágenes al servidor y recibir la respuesta. En entornos con mala cobertura o conexión inestable, el rendimiento del sistema se resiente, introduciendo retrasos o incluso imposibilitando el reconocimiento.

Además de las limitaciones técnicas del ESP32-CAM, también hay que tener en cuenta las propias de un proyecto académico. El tiempo para desarrollarlo es limitado y no se busca crear un sistema comercial completamente terminado. Por eso, en esta primera versión no se han incluido funciones como la conexión directa a cerraduras

electrónicas o el uso de cifrado avanzado en las comunicaciones. Son aspectos que podrían añadirse más adelante, pero que ahora quedarían fuera del alcance.

En resumen, estas limitaciones definen el marco de trabajo del TFG. Aunque reducen las posibilidades de implementación, ayudan a enfocarse en lo esencial: mostrar que un sistema de control de acceso facial puede funcionar de manera fiable con hardware económico y software libre.

Capítulo 4

Recursos

El desarrollo de este TFG ha requerido una serie de recursos, tanto humanos como técnicos, que han permitido implementar el sistema propuesto y realizar las pruebas necesarias. Los recursos seleccionados se caracterizan por ser de bajo coste y de fácil acceso, lo que facilita la replicabilidad del proyecto. Además, gran parte del software empleado es de código abierto, reforzando el carácter académico y la filosofía de accesibilidad que motivan este TFG.

4.1. Recursos humanos

El proyecto ha sido desarrollado de manera individual como parte del Trabajo Fin de Grado. Se ha contado con la supervisión del tutor académico, que ha prestado apoyo en la resolución de dudas y problemas surgidos durante el desarrollo.

Además, ha sido de gran utilidad la documentación oficial de los fabricantes y el soporte disponible en comunidades de desarrolladores, especialmente en relación con el ESP32-CAM, FastAPI e InsightFace.

4.2. Recursos hardware

Los elementos hardware empleados en el proyecto han sido los siguientes:

- ESP32-CAM: módulo central del sistema, que combina microcontrolador y cámara integrada.

- Ordenador personal: utilizado para programar el ESP32-CAM, ejecutar el servidor backend y analizar los resultados de las pruebas.
- Red Wi-Fi: imprescindible para la comunicación entre el ESP32-CAM y el servidor.
- Accesorios: adaptador USB para la programación inicial del ESP32-CAM y cableado auxiliar.

4.3. Recursos software

En el plano software, el proyecto ha combinado herramientas para el desarrollo del firmware, el backend y las interfaces de usuario:

- Arduino IDE: entorno usado para programar el ESP32-CAM.
- ESP-WHO: librería optimizada para el reconocimiento facial local en el ESP32.
- FastAPI: framework en Python elegido para el desarrollo del backend.
- InsightFace: librería de aprendizaje profundo utilizada para el reconocimiento facial en el servidor.
- Motor de base de datos ligero (SQLite): base de datos ligera empleada para el almacenamiento de registros.
- Flutter + Android Studio: utilizados para el desarrollo y pruebas de la aplicación móvil, que se ejecutó en el emulador del IDE.
- Visual Studio Code: editor de código empleado para el backend y la aplicación móvil.
- ChatGPT: herramienta de apoyo basada en inteligencia artificial, utilizada como asistente para resolver dudas técnicas puntuales y generar ejemplos de código que posteriormente fueron revisados, adaptados y validados en el desarrollo del proyecto.
- GitHub: utilizado como repositorio y herramienta de control de versiones. El código fuente completo desarrollado para este Trabajo de Fin de Grado está disponible en: **Repositorio en GitHub**.

Capítulo 5

Especificación de Requisitos

La correcta definición de requisitos es un paso fundamental en cualquier proyecto, ya que establece con claridad qué funciones debe cubrir el sistema y cuáles son las condiciones bajo las que debe hacerlo. En este TFG los requisitos se dividen en funcionales y no funcionales.

5.1. Requisitos funcionales

El sistema debe permitir:

1. Capturar imágenes de los rostros de los usuarios de forma rápida y con calidad suficiente para su posterior procesamiento.
2. Identificar y diferenciar de manera fiable entre rostros registrados y no registrados.
3. Ofrecer dos modos de procesamiento:
   - **Modo local:** el reconocimiento se ejecuta íntegramente en el ESP32-CAM.
   - **Modo remoto:** el dispositivo envía la imagen al servidor, que realiza el procesamiento.
4. Enviar y recibir datos entre el dispositivo y el servidor mediante una comunicación estable y segura.
5. Permitir la gestión de usuarios (añadir, eliminar y consultar los registrados) de forma accesible.

6. Registrar cada evento de acceso, almacenando información como fecha, hora y usuario identificado.
7. Manejar correctamente los intentos de acceso de personas no registradas, notificando el resultado sin provocar fallos en el sistema.

5.2. Requisitos no funcionales

Además, el sistema debe cumplir las siguientes condiciones:

1. Ser una solución económica, basada en hardware accesible y de bajo coste.
2. Mantener un consumo energético reducido en el dispositivo local, permitiendo su uso continuado.
3. Estar diseñado para ser escalable, de forma que pueda ampliarse el número de usuarios y registros sin modificaciones profundas.
4. Utilizar librerías, frameworks y herramientas de código abierto o de acceso gratuito, facilitando la replicación, el mantenimiento y la actualización del sistema sin costes añadidos.
5. Ofrecer una interacción intuitiva y sencilla, de modo que cualquier usuario pueda operar el sistema sin necesidad de conocimientos técnicos avanzados.

Capítulo 6

Especificación del Sistema

6.1. Descripción general

El sistema de control de acceso planteado se diseñó inicialmente para funcionar en modo local sobre el ESP32-CAM, aprovechando su cámara integrada y la librería ESP-WHO para la detección facial. Sin embargo, pronto se evidenciaron las limitaciones del dispositivo en cuanto a memoria y potencia de cálculo, que impedían cargar modelos de reconocimiento preentrenados o manejar múltiples usuarios de manera eficiente.

Por ello, el diseño evolucionó hacia una arquitectura híbrida con dos modos de funcionamiento:

- **Modo local:** el ESP32-CAM captura y procesa imágenes de forma autónoma, generando embeddings faciales y comparándolos con los almacenados en el propio dispositivo.
- **Modo remoto:** el ESP32-CAM actúa como dispositivo de captura y envía las imágenes al servidor backend, que dispone de mayor capacidad de cálculo y se encarga tanto del reconocimiento facial como de la gestión centralizada de usuarios y registros.

De esta manera, el sistema combina la rapidez y autonomía del modo local con la precisión y escalabilidad del modo remoto, ofreciendo flexibilidad según las necesidades de uso.

6.2. Arquitectura del sistema

El sistema se estructura en cuatro bloques principales:

- **Dispositivo ESP32-CAM:** encargado de capturar imágenes en tiempo real. En modo local realiza detección y reconocimiento básico.
- **Servidor backend (FastAPI + InsightFace + SQLite):** procesa las imágenes en el modo remoto, gestiona usuarios y almacena registros.
- **Almacenamiento:** embeddings faciales guardados en archivos binarios para un acceso rápido y registros en una base de datos SQLite.
- **Interfaces de usuario:** dos alternativas de interacción:
  - Una interfaz web ligera servida por el ESP32-CAM.
  - Una aplicación móvil multiplataforma desarrollada en Flutter.

La comunicación entre los distintos componentes se realiza mediante HyperText Transfer Protocol (HTTP)/JavaScript Object Notation (JSON) para operaciones puntuales y *WebSocket* para actualizaciones en tiempo real.

6.3. Flujo de datos

El sistema contempla dos modos de funcionamiento: local (procesamiento en el ESP32-CAM) y remoto (procesamiento en el servidor backend). Ambos permiten realizar las mismas operaciones fundamentales (registro, reconocimiento y eliminación de usuarios), aunque varía el lugar donde se ejecuta el procesamiento y dónde se almacenan los datos.

6.3.1. Modo local (procesamiento en ESP32-CAM)

1. El ESP32-CAM obtiene la imagen en tiempo real mediante su cámara integrada.
2. El dispositivo carga en memoria los embeddings de los usuarios registrados (si existen).

3. **Registrar usuario:** se generan embeddings a partir de varias capturas del rostro detectado y se guardan en un archivo binario local independiente al modo remoto.
4. **Reconocer usuario:** se genera un embedding de la captura actual y se compara con los almacenados en memoria.
5. **Eliminar usuarios:** se borra el embedding individual de un usuario o el conjunto completo de usuarios registrados.
6. El ESP32-CAM genera un mensaje de confirmación:
   - Si se usa la app móvil, la confirmación se envía al servidor y este la reenvía a la aplicación.
   - Si se usa la interfaz web, la confirmación se transmite directamente al navegador desde el dispositivo.
7. Los registros de accesos se almacenan y se visualizan a través de la aplicación móvil.

6.3.2. Modo remoto (procesamiento en servidor)

1. El ESP32-CAM captura la imagen en tiempo real y la envía al servidor backend.
2. El servidor procesa la imagen con InsightFace, que detecta el rostro y permite:
   - **Registrar usuario:** genera embeddings a partir de varias capturas y los guarda en un archivo binario específico para el modo remoto.
   - **Reconocer usuario:** genera un embedding de la captura actual y lo compara con los ya almacenados.
   - **Eliminar usuarios:** permite borrar un usuario concreto o todos los registrados.
3. Una vez realizada la operación, el servidor envía un mensaje de confirmación:
   - A la aplicación móvil, notificando el resultado.
   - A la interfaz web, a través del ESP32.
4. Los registros de accesos quedan almacenados en la base de datos SQLite y se consultan desde la aplicación móvil.

6.4. Modelos de datos

El sistema emplea un modelo de datos deliberadamente sencillo, suficiente para cubrir las necesidades de persistencia y consulta de la información generada durante su funcionamiento.

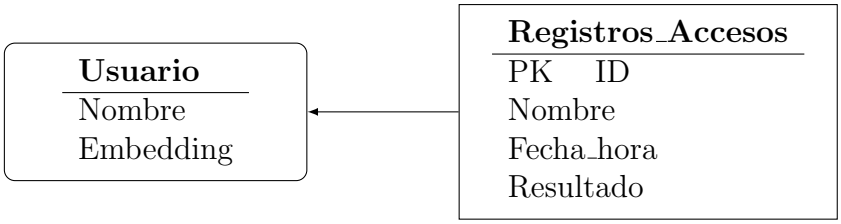

Figura 6.1: Modelo de datos del sistema de control de accesos.

Se distinguen dos entidades principales:

- **Usuarios:** representan a las personas registradas. Cada usuario se identifica mediante un nombre único y sus embeddings faciales se almacenan en archivos binarios independientes para cada modo de procesamiento (local y remoto). De este modo, se evita sobrecargar la base de datos con vectores de alta dimensión y se garantiza un acceso rápido. Aunque no forman parte directamente de la base de datos relacional, son esenciales dentro del modelo de información.
- **Registros de acceso:** se gestionan en una base de datos SQLite. Cada registro contiene un identificador único, el nombre del usuario reconocido (o desconocido), la fecha y hora del intento y el resultado de la operación (concedido o denegado). Esto permite mantener un historial ordenado, persistente y fácilmente consultable.

La relación entre ambas entidades es de uno a muchos, ya que un mismo usuario puede generar múltiples registros de acceso a lo largo del tiempo. Este modelo híbrido (con embeddings en archivos binarios y registros en SQLite) proporciona un equilibrio adecuado entre simplicidad, rendimiento y trazabilidad.

6.5. Modelos funcionales

Para completar la arquitectura y los flujos de datos descritos anteriormente, se elaboraron modelos funcionales que representan gráficamente las interacciones entre los componentes del sistema y el usuario. Estos esquemas permiten visualizar con

claridad las funciones que desempeña cada módulo y cómo se coordinan entre sí, facilitando la comprensión global del sistema.

6.5.1. Modelo funcional con interfaz móvil

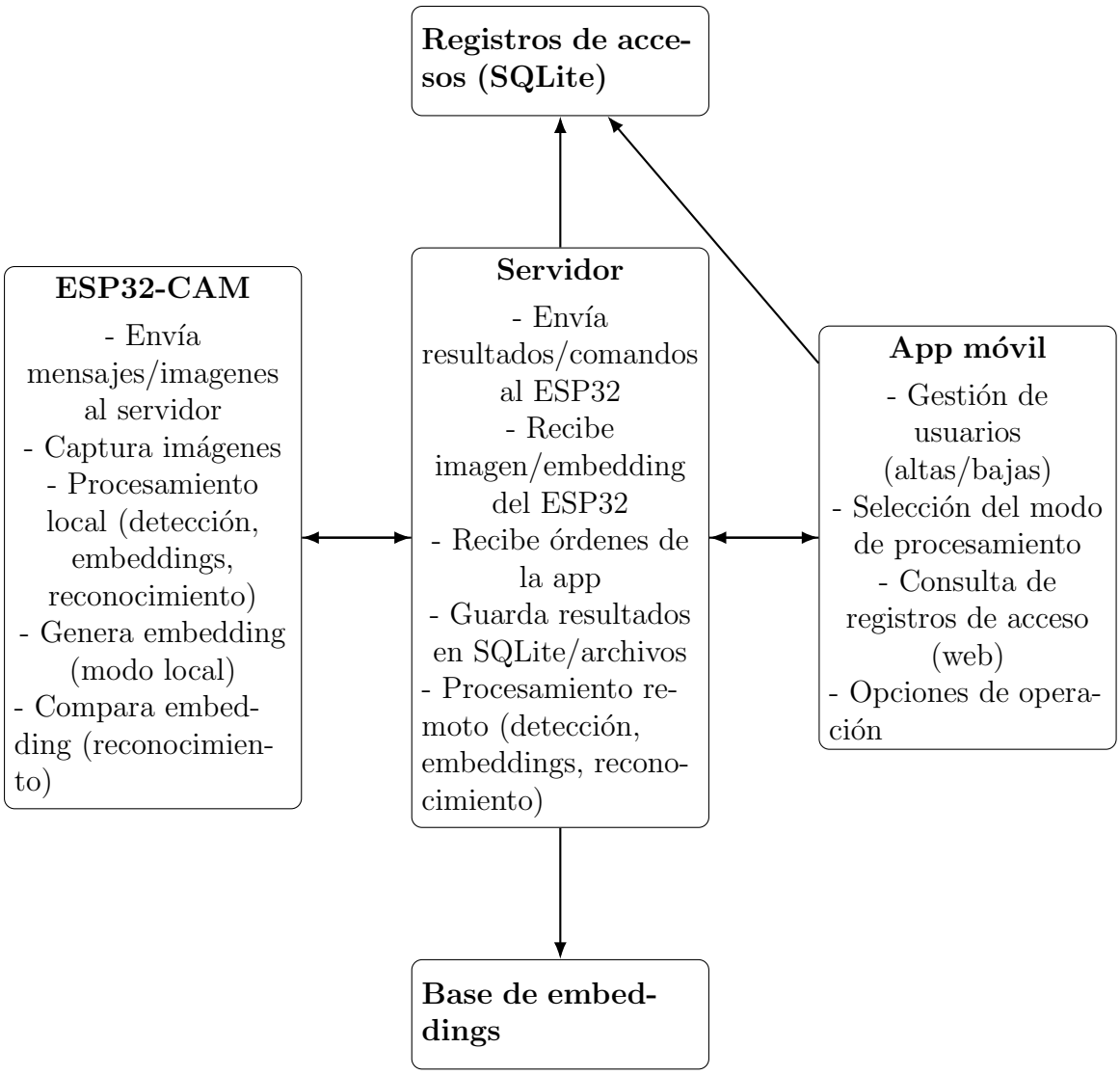

Figura 6.2: Modelo funcional del sistema con aplicación móvil.

En esta versión, la aplicación móvil es el centro de interacción del usuario y coordina el funcionamiento del sistema:

- ESP32-CAM: captura imágenes y puede realizar el procesamiento local mediante ESP-WHO. En modo remoto, envía las imágenes o embeddings al servidor y recibe de este las instrucciones y resultados.
- **Servidor:** actúa como núcleo del sistema en el modo remoto. Gestiona órdenes provenientes de la app móvil, recibe imágenes o embeddings desde el ESP32-CAM y ejecuta el procesamiento facial con InsightFace. Los resultados se almacenan en la base de datos y los embeddings en archivos binarios.
- **Aplicación móvil (Flutter):** centraliza la interacción con el usuario. Permite gestionar usuarios (altas y bajas), seleccionar el modo de procesamiento (local o remoto), consultar los registros de acceso y ejecutar operaciones de detección y reconocimiento.
- **Base de datos y registros:** el servidor mantiene conexión con el almacén de embeddings y con la base SQLite, garantizando la persistencia y trazabilidad de la información.

6.5.2. Modelo funcional con interfaz web

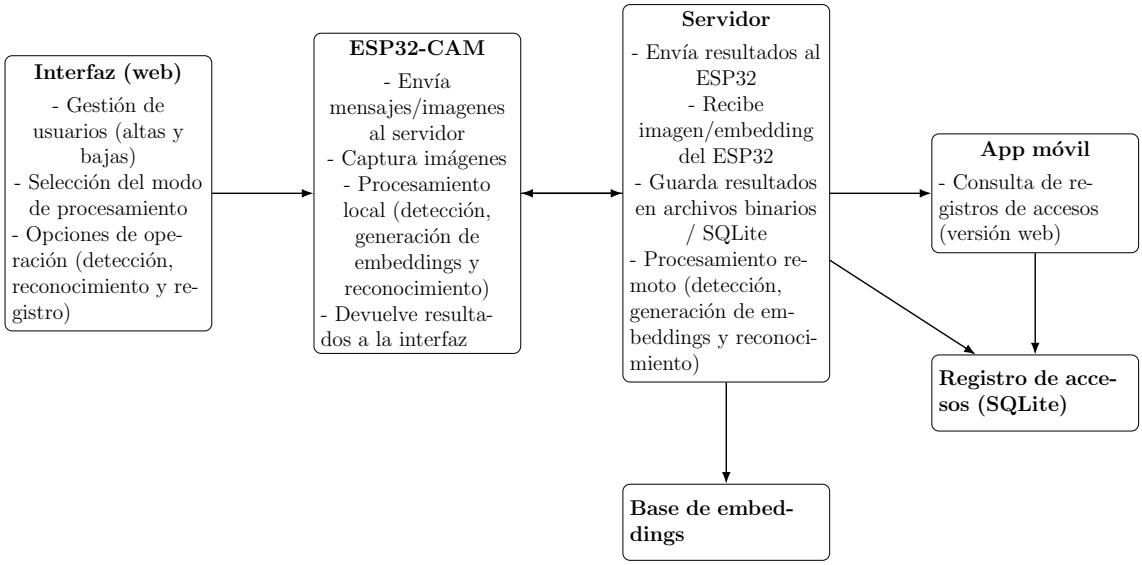

Figura 6.3: Modelo funcional del sistema con interfaz web.

En esta segunda versión, el ESP32-CAM ofrece un funcionamiento más autónomo, sirviendo la interfaz directamente al navegador:

- ESP32-CAM: además de capturar imágenes, realiza el procesamiento local con ESP-WHO. Hospeda la interfaz web y actúa como servidor mediante WebSocket, enviando notificaciones y resultados inmediatos al navegador.
- **Interfaz web:** accesible desde cualquier navegador de la red local, permite visualizar el streaming en directo, seleccionar el modo de procesamiento y gestionar usuarios. La comunicación con el ESP32 se realiza vía WebSocket.
- **Servidor:** desempeña un papel secundario, recibiendo imágenes o embeddings del ESP32, almacenando/cargando embeddings cuando es necesario y guardando registros de acceso en SQLite.
- **Aplicación móvil:** en este modelo también pasa a un rol secundario, limitado a la consulta de registros de accesos almacenados en la base de datos.

En este escenario, la mayor parte del procesamiento se concentra en el ESP32-CAM, lo que facilita un funcionamiento autónomo del sistema. No obstante, las limitaciones de memoria y capacidad de cómputo restringen la escalabilidad en comparación con el modelo basado en la aplicación móvil y el servidor como núcleo.

6.6. Desarrollo de los componentes

El sistema desarrollado se compone de distintos elementos hardware y software que interactúan para dar soporte a las funciones de captura, procesamiento y gestión de usuarios. A continuación se detallan los componentes principales, sus características y el papel que desempeñan en la solución.

6.6.1. ESP32-CAM

El dispositivo central de captura es el ESP32-CAM, un microcontrolador de la familia ESP32 desarrollado por Espressif Systems. Su principal diferencia respecto a otros modelos de la misma familia es la integración de una cámara y un conector microSD, lo que reduce la necesidad de periféricos externos y simplifica el diseño del sistema.

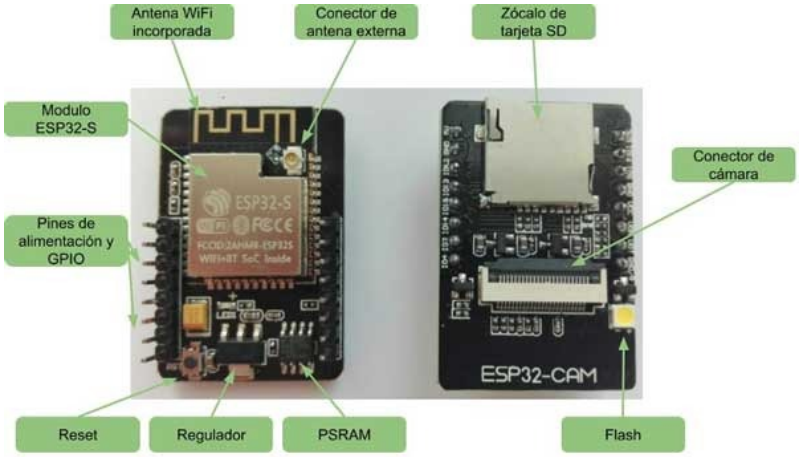

Figura 6.4: Componentes principales del módulo ESP32-CAM

Especificaciones técnicas relevantes:

- **Módulo ESP32-S:** incluye CPU de 32 bits de bajo consumo con dos núcleos, hasta 160 MHz de velocidad de reloj.
- **Memoria:** 520 KB de Static Random Access Memory (SRAM) interna y 4 MB de Pseudo Static RAM (PSRAM) externa, lo que condiciona la cantidad de embeddings que pueden manejarse en local.
- **Conectividad inalámbrica:**
  - Wi-Fi 802.11 b/g/n/e/i, con protocolos de seguridad WPA/WPA2/WPA2-Enterprise/WPS.
  - Bluetooth 4.2 con soporte BLE.
- **Interfaces:** UART, SPI, I2C y PWM, con hasta 9 pines General Purpose Input/Output (GPIO) disponibles.
- **Almacenamiento externo:** soporte para tarjetas microSD de hasta 4 GB.
- **Modos de reposo (sleep modes):**
  - Deep-sleep: 6 mA @ 5V.
  - Light-sleep: 6.7 mA @ 5V.
  - Modem-sleep: hasta 20 mA @ 5V.
- **Firmware Over-The-Air (actualización remota) (OTA):** soporta actualización remota del firmware a través de Over-The-Air.

- **Cámara:**
  - Compatible con sensores OV2640 y OV7670.
  - En este proyecto se utilizó el OV2640, de 2 megapíxeles.
  - Resolución máxima UXGA (1622×1200 px).
  - Soporte de formatos JPEG (nativo en OV2640), BMP y escala de grises.
  - Velocidad de captura de entre 15 y 60 FPS, dependiendo de la resolución seleccionada.
- **Alimentación:** funciona a 5 V, con posibilidad de usar antena externa en caso de redes inestables.

Estas características hacen del ESP32-CAM una plataforma adecuada para prototipos de visión artificial por su bajo coste y versatilidad, aunque limitada en términos de memoria y capacidad de cómputo, lo que condicionó el diseño híbrido del sistema [15].

6.6.1.1. ESP-WHO

Para dotar al ESP32-CAM de capacidades de visión por computador se empleó ESP-WHO, una plataforma oficial de Espressif orientada a la detección y reconocimiento facial en dispositivos de esta familia. Esta librería se apoya en ESP-DL, un framework ligero de inferencia de redes neuronales optimizado para el hardware de Espressif.

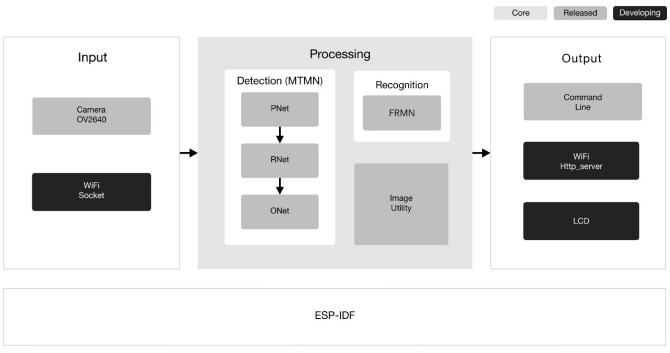

Figura 6.5: Arquitectura del framework ESP-WHO

El flujo de trabajo de ESP-WHO comienza con una fase de preprocesamiento de la imagen, en la que se realizan operaciones habituales en visión artificial como el

recorte de la región de interés, el escalado a la resolución requerida, la normalización de valores de píxeles y la conversión de formatos. Estas transformaciones permiten que los modelos ligeros que se ejecutan en el microcontrolador trabajen de manera más eficiente en un hardware con recursos limitados.

A continuación, entra en juego el módulo MTMN, que combina las arquitecturas MTCNN (Multi-Task Cascaded Convolutional Networks) y MobileNet. Este bloque se encarga de detectar los rostros presentes en la imagen de entrada, proporcionando sus coordenadas y puntos clave faciales, lo que permite recortar y preparar las regiones relevantes para el reconocimiento posterior.

El siguiente paso es el módulo MobileFace, responsable de generar los embeddings faciales. El proceso consiste en identificar puntos clave del rostro (ojos, nariz, comisuras de la boca) para alinear la imagen y corregir posibles inclinaciones. Posteriormente, el rostro se recorta y se redimensiona al tamaño requerido por el modelo, y finalmente se obtiene un vector numérico que representa de manera compacta la identidad facial.

En este proyecto, ESP-WHO resultó esencial para habilitar el modo de funcionamiento local, ofreciendo un sistema autónomo de reconocimiento facial. No obstante, las limitaciones inherentes al hardware hicieron evidente que este modo debía complementarse con un procesamiento remoto más potente para mejorar precisión y escalabilidad [27].

6.6.1.2. Arduino IDE

El desarrollo del firmware del ESP32-CAM se llevó a cabo en Arduino IDE, un entorno de programación ampliamente utilizado para microcontroladores. Su principal ventaja es la facilidad de uso y la amplia disponibilidad de librerías, además de ofrecer compatibilidad oficial con la familia ESP32 y soporte para ESP-WHO.

Dentro de este proyecto, Arduino IDE permitió inicializar la cámara, capturar y preprocesar imágenes, ejecutar el flujo completo de detección y reconocimiento en local y gestionar usuarios en memoria o archivos binarios. También se programaron las comunicaciones con el servidor mediante HTTP y WebSocket y la carga de una interfaz web básica directamente en el dispositivo [19].

6.6.2. Servidor backend

El servidor se diseñó para complementar al ESP32-CAM, ejecutando el reconocimiento avanzado en modo remoto y centralizando la gestión de usuarios y registros.

6.6.2.1. FastAPI

Para implementar el backend se utilizó FastAPI, un framework moderno en Python pensado para construir APIs rápidas y con soporte nativo para programación asíncrona. En este proyecto se empleó para recibir imágenes enviadas por el ESP32-CAM, ejecutar el reconocimiento facial en remoto, gestionar los embeddings y almacenar los registros de acceso en la base de datos SQLite. FastAPI también facilita el desarrollo gracias a la generación automática de documentación de los endpoints a través de Swagger UI, lo que resultó muy útil en las fases de prueba [24].

6.6.2.2. SQLite

La base de datos utilizada fue SQLite, seleccionada por su sencillez y ligereza. Al trabajar sobre un único archivo en disco, no requiere de un servidor de bases de datos independiente, lo que facilita su integración en prototipos. En este sistema se utilizó para almacenar el historial de accesos con información sobre el identificador, la fecha, la hora, el usuario reconocido y el resultado de la operación [28].

6.6.2.3. InsightFace

El motor de reconocimiento facial en modo remoto se implementó con InsightFace, un framework de análisis facial en 2D y 3D desarrollado en torno a PyTorch y MXNet. Se seleccionó el modelo `buffalo_1`, basado en una arquitectura ResNet optimizada que genera embeddings de 512 dimensiones.

El flujo de funcionamiento comienza con la detección del rostro en la imagen, continúa con la localización de puntos clave (ojos, nariz, boca) y la normalización de la cara, y finalmente la red procesa la imagen a través de sus capas convolucionales con "skip connections". Estas conexiones residuales permiten que la entrada de un bloque se sume a su salida, lo que evita el problema del desvanecimiento del gradiente y facilita el entrenamiento de redes más profundas y precisas.

El resultado es un embedding que representa al rostro en un espacio vectorial. Dicho embedding se compara con los almacenados mediante producto escalar para calcular similitudes y tomar decisiones de reconocimiento.

InsightFace permitió alcanzar un nivel de precisión y escalabilidad superior al modo local del ESP32-CAM, reforzando el valor de la arquitectura híbrida planteada en este TFG [17, 18, 29].

6.6.3. Interfaces de usuario

El sistema ofrece dos alternativas de interacción: una interfaz web ligera servida directamente por el ESP32-CAM y una aplicación móvil multiplataforma desarrollada en Flutter. Ambas cumplen con el requisito de proporcionar una interacción intuitiva y accesible para la gestión de usuarios y la consulta de accesos.

6.6.3.1. Interfaz web

La interfaz web se desarrolló en HTML, CSS y JavaScript y es servida directamente por el ESP32-CAM. Esto permite que cualquier usuario pueda acceder desde un navegador dentro de la red local, sin necesidad de instalar software adicional.

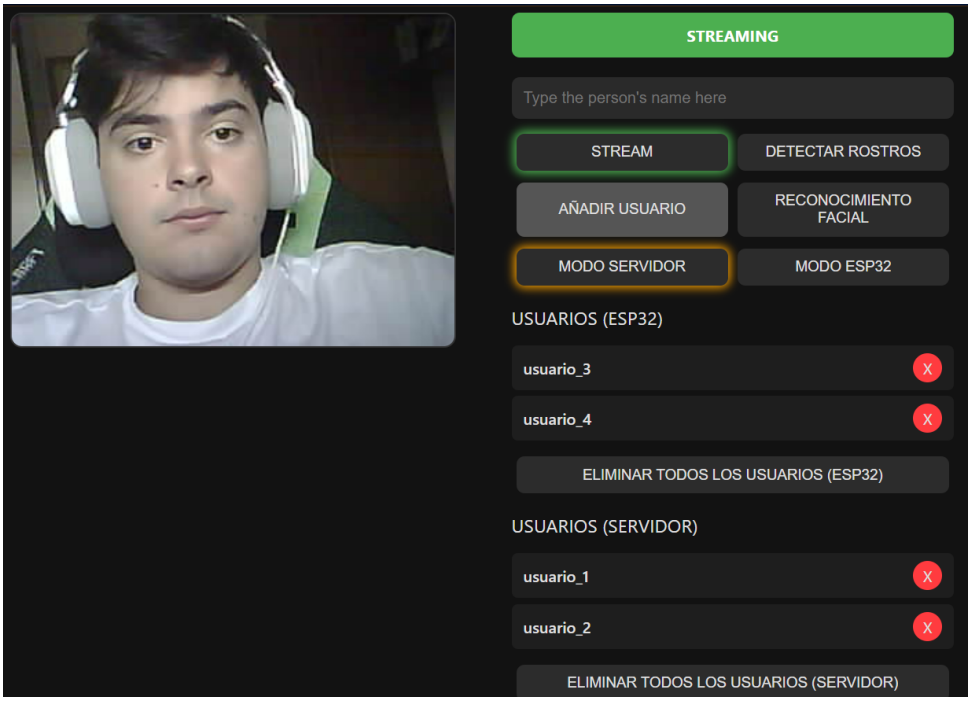

Figura 6.6: Interfaz web servida por el ESP32-CAM

La pantalla principal se diseñó con una estructura sencilla: en la zona izquierda se mostraba el flujo de vídeo en tiempo real capturado por la cámara, mientras que en la parte derecha se concentraban los controles de operación y las funciones de gestión de usuarios. La comunicación entre el navegador y el dispositivo se realizaba a través de *WebSocket*, lo que permitía recibir notificaciones inmediatas sobre eventos como detecciones, reconocimientos o confirmaciones de operaciones.

Entre las funciones disponibles se encontraba la posibilidad de alternar entre el modo local (procesamiento en el propio ESP32-CAM) y el modo remoto (procesamiento en servidor), activar la detección de rostros, iniciar el reconocimiento facial comparando contra los usuarios registrados o añadir nuevos usuarios introduciendo su nombre y generando los embeddings correspondientes desde la cámara. Asimismo, la interfaz permitía consultar los usuarios existentes y eliminarlos, ya fuese de manera individual o de forma global.

En definitiva, esta solución cumplía con el objetivo de ofrecer un entorno ligero y accesible para el uso básico del sistema, resultando especialmente útil en entornos sin acceso al backend.

6.6.3.2. Aplicación móvil

Con el fin de ofrecer una experiencia más completa y centralizada se desarrolló una aplicación móvil en Flutter, empleando el lenguaje Dart. Su ejecución se probó en el emulador de Android Studio y, a diferencia de la interfaz web, esta versión se comunicaba directamente con el servidor backend tanto por HTTP (para operaciones puntuales como el registro de usuarios o la consulta de datos) como por *WebSocket* (para actualizaciones en tiempo real de accesos y resultados de reconocimiento).

Pantallas principales:

- **Historial de accesos:** se muestran en formato de lista todos los registros almacenados en SQLite. Cada entrada incluye fecha, hora, usuario reconocido (o "desconocido" en caso contrario) y el resultado del acceso. Desde esta pantalla es posible navegar hacia las demás secciones.

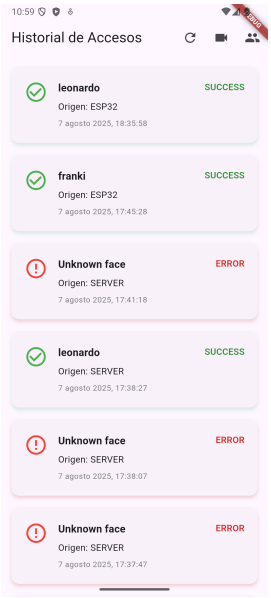

Figura 6.7: Pantalla de historial de accesos en la aplicación móvil.

- **Transmisión de vídeo en tiempo real:** se visualiza la señal de la cámara del ESP32-CAM y se dispone de controles para alternar entre los modos local y remoto, activar la detección, realizar el reconocimiento o registrar nuevos usuarios.

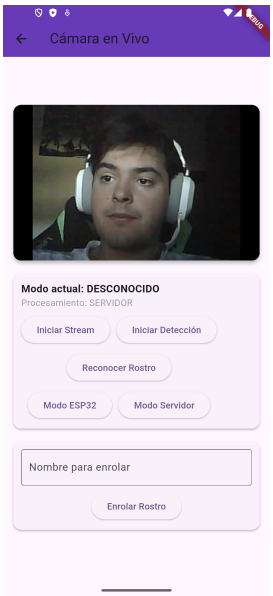

Figura 6.8: Pantalla de transmisión en vivo desde la cámara del ESP32-CAM.

- **Gestión de usuarios:** muestra el listado de usuarios registrados junto con opciones para añadir o eliminar usuarios de forma centralizada, ya sea de manera individual o eliminando la base completa.

Figura 6.9: Pantalla de gestión de usuarios en la aplicación móvil.

En conclusión, la aplicación cumple el requisito de ser la interfaz principal para la administración, proporcionando una experiencia más completa que la web y garantizando la portabilidad al poder ejecutarse en dispositivos Android e iOS.

6.7. Pruebas y resultados

Con el fin de validar el sistema en relación con los requisitos planteados, se diseñó una batería de pruebas que abordaron tanto las funciones esenciales como el rendimiento y la capacidad de respuesta en condiciones adversas. Los resultados obtenidos permiten comparar los dos modos de procesamiento (ESP32-CAM y servidor remoto) y valorar sus ventajas y limitaciones.

6.7.1. Métricas de evaluación

Para evaluar el rendimiento del sistema se emplearon principalmente las métricas de exactitud (*accuracy*) y precisión (*precision*). Adicionalmente, se mencionan otras

métricas estándar en el ámbito del reconocimiento facial, como *recall* y F1-score, aunque no se calcularon en este trabajo.

Exactitud (Accuracy):

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.1}$$

Precisión (Precision):

$$Precision = \frac{TP}{TP + FP} \tag{6.2}$$

Exhaustividad o Sensibilidad (Recall):

$$Recall = \frac{TP}{TP + FN} \tag{6.3}$$

Medida F1 (F1-score):

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{6.4}$$

En las pruebas realizadas, los valores de falsos positivos (FP) y falsos negativos (FN) fueron prácticamente nulos, lo que explica que los resultados de exactitud y precisión se aproximen al 100 %. Este resultado debe interpretarse con cautela, ya que responde a la baja variabilidad del conjunto de pruebas. En escenarios con mayor número de usuarios y condiciones menos controladas es esperable que aparezcan errores y que estas métricas se reduzcan.

Aunque *recall* y F1-score no se emplearon en este TFG, se reconocen como indicadores más representativos en pruebas con mayor variabilidad y se consideran relevantes para futuras evaluaciones.

6.7.2. Pruebas funcionales

Estas pruebas se centraron en verificar que las funcionalidades básicas del sistema se ejecutan correctamente:

- **Captura de imágenes:** el ESP32-CAM generó un flujo estable de 15–20 FPS a resoluciones moderadas, garantizando la base para la detección facial en tiempo real.

- **Reconocimiento de usuarios registrados:** ambos modos lograron una identificación correcta en condiciones normales (30–40 cm, rostro frontal), alcanzando una exactitud y precisión del 100 %.
- **Gestión de usuarios:** las operaciones de alta, baja y consulta se realizaron correctamente tanto desde la interfaz web como desde la aplicación móvil, con actualizaciones coherentes en los archivos binarios (embeddings) y en la base de datos.
- **Registro de accesos:** todos los intentos quedaron almacenados en SQLite con los campos de fecha, hora, usuario reconocido y resultado de la operación.
- **Usuarios no registrados:** en estos casos el sistema devolvió el resultado "desconocido" y denegó el acceso, sin fallos ni bloqueos.

El sistema cumple con los requisitos esenciales, garantizando captura, reconocimiento, gestión de usuarios y trazabilidad de accesos.

6.7.3. Pruebas de rendimiento

El análisis de rendimiento tuvo como objetivo medir la velocidad de respuesta y la estabilidad temporal en ambos modos de procesamiento. En condiciones normales de uso (rostro frontal a 30–40 cm), los resultados fueron los siguientes:

Tabla 6.1: Resultados de rendimiento en condiciones normales (rostro frontal a 30–40 cm).

Modo	TP	Errores	FP	FN	Exactitud (%)	Tiempo medio (ms)
ESP32	100	0	0	0	100,00	848.86
Servidor	100	0	0	0	100,00	878.10

En condiciones normales, ambos modos alcanzaron el 100 % de exactitud y precisión. El ESP32-CAM presentó un tiempo medio de 848,86 ms estable, mientras que el servidor obtuvo un tiempo medio de 878,10 ms pero con mayor variabilidad, incluyendo picos superiores a 2000 ms por dependencia de la red y la carga del sistema.

6.7.4. Pruebas de robustez

Estas pruebas analizaron el comportamiento del sistema en condiciones adversas: distancia, ángulo, iluminación, múltiples rostros, accesorios y movimiento. Para cada prueba se realizaron 20 ejecuciones.

Distancia

Tabla 6.2: Resultados de robustez en función de la distancia al sensor.

Condición	Modo	TP	Errores	FP	FN	Exactitud (%)	Precisión (%)	Tiempo medio (ms)
<20 cm	ESP32	20	0	0	0	100,00	100,00	969.80
	Servidor	20	0	0	0	100,00	100,00	696.50
30–40 cm	ESP32	20	0	0	0	100,00	100,00	818.55
	Servidor	20	0	0	0	100,00	100,00	882.65
1 m	ESP32	19	1	0	0	95,00	100,00	810.05
	Servidor	20	0	0	0	100,00	100,00	785.45
2 m	ESP32	0	0	0	0	0,00	0,00	0
	Servidor	0	20	0	0	0,00	0,00	0

El ESP32-CAM mostró un rendimiento muy sólido a corta distancia: tanto a menos de 20 cm como a 30–40 cm, logró un 100 % de exactitud y precisión, con tiempos de respuesta estables en torno a 800–950 ms. A 1 metro la tasa de aciertos se redujo ligeramente (95 %), aunque los tiempos se mantuvieron estables. A partir de los 2 metros, el dispositivo ya no logró detectar rostros.

El servidor, por su parte, mantuvo un reconocimiento perfecto hasta 1 metro, con tiempos de respuesta comparables a los del ESP32-CAM. A 2 metros, aunque fue capaz de detectar rostros, no logró reconocerlos correctamente. En conjunto, ambos sistemas resultan adecuados en escenarios de corta y media distancia, pero muestran limitaciones en entornos donde el usuario se encuentra demasiado alejado del sensor.

Ángulo de captura

Tabla 6.3: Resultados de robustez en función del ángulo de captura.

Condición	Modo	TP	Errores	FP	FN	Exactitud (%)	Precisión (%)	Tiempo medio (ms)
45º izquierda	ESP32	0	6	0	0	0,00	0,00	836.00
	Servidor	20	0	0	0	100,00	100,00	1823.15
90º izquierda	ESP32	0	6	0	0	0,00	0,00	836.00
	Servidor	20	0	0	0	100,00	100,00	3340.85
45º derecha	ESP32	0	6	0	0	0,00	0,00	836.00
	Servidor	20	0	0	0	100,00	100,00	1719.85
90º derecha	ESP32	0	6	0	0	0,00	0,00	836.00
	Servidor	20	0	0	0	100,00	100,00	1663.50
45º hacia abajo	ESP32	0	20	0	0	0,00	0,00	980.55
	Servidor	0	20	0	0	0,00	0,00	1009.20
45º hacia arriba	ESP32	0	0	0	0	0,00	0,00	0
	Servidor	4	16	0	0	20,00	100,00	1003.85

En este escenario las diferencias fueron mucho más notables. El ESP32-CAM no consiguió reconocer rostros con desviaciones de 45° ni de 90°, e incluso en la mayoría de ocasiones no llegó a detectarlos. Por tanto, su efectividad se restringe prácticamente a situaciones con rostros frontales.

En contraste, el servidor mantuvo un 100 % de aciertos incluso en ángulos pronunciados, lo que evidencia su mayor robustez. No obstante, esta capacidad tuvo un coste en rendimiento: los tiempos de respuesta se incrementaron de manera significativa, alcanzando hasta 3,3 segundos en desviaciones de 90°.

Iluminación

Tabla 6.4: Resultados de robustez en condiciones de iluminación.

Condición	Modo	TP	Errores	FP	FN	Exactitud (%)	Precisión (%)	Tiempo medio (ms)
Luz no natural arriba	ESP32	9	11	0	0	45,00	100,00	743.05
	Servidor	2	18	0	0	10,00	100,00	1798.25
Foco directo artificial	ESP32	18	1	1	0	90,00	94,74	962.32
	Servidor	20	0	0	0	100,00	100,00	1026.25
Luz tenue (monitor)	ESP32	16	4	3	0	80,00	84,21	925.55
	Servidor	20	0	0	0	100,00	100,00	810.00
Contraluz desde atrás	ESP32	15	5	0	0	75,00	100,00	918.45
	Servidor	–	–	–	–	–	–	–

Las condiciones lumínicas introdujeron resultados interesantes. Bajo un foco directo, ambos modos mantuvieron un alto rendimiento (90 % en el ESP32-CAM frente al 100 % en el servidor). En situaciones de luz tenue, el ESP32-CAM redujo su exactitud al 80 %, mientras que el servidor se mantuvo en 100 %.

Sin embargo, en escenarios de contraluz y con luz procedente de arriba, el ESP32-CAM obtuvo más aciertos que el servidor, mostrando un comportamiento algo más estable en esos casos concretos. En general, el servidor volvió a ser más consistente en el conjunto de pruebas, manteniendo exactitudes cercanas al 100 %, aunque en determinadas condiciones el ESP32-CAM logró superarlo puntualmente.

Múltiples rostros

Tabla 6.5: Resultados de robustez en escenas con múltiples rostros.

Condición	Modo	TP	Errores	FP	FN	Exactitud (%)	Precisión (%)	Tiempo medio (ms)
Dos rostros (1 registrado)	ESP32	12	8	0	0	60,00	100,00	927.95
	Servidor	6	14	0	0	30,00	100,00	1135.50
Dos rostros (2 registrados)	ESP32	20	0	0	0	100,00	100,00	906.50
	Servidor	20	0	0	0	100,00	100,00	1103.50

La presencia de varios usuarios en la escena representó un reto adicional. En el caso de que hubiese dos rostros y solo uno estuviera registrado, el sistema alternó en algunos intentos el rostro detectado, lo que redujo la tasa de aciertos medida.

Sin embargo, es importante matizar que en ningún momento se produjo un falso positivo: el usuario no registrado nunca fue identificado como válido, sino que el sistema simplemente no priorizó siempre el mismo rostro del encuadre.

Cuando ambos rostros estaban registrados, tanto el ESP32-CAM como el servidor lograron un 100 % de aciertos, aunque intercalando el reconocimiento de ambos usuarios en distintos intentos. Es decir, el sistema fue capaz de reconocer correctamente a todos los usuarios presentes, lo que confirma que la limitación no está en el algoritmo de reconocimiento facial, sino en la gestión de escenas multiusuario, donde sería deseable contar con un mecanismo de seguimiento de rostros para fijar la atención en un sujeto concreto.

Accesorios en el rostro

Tabla 6.6: Resultados de robustez con accesorios en el rostro.

Condición	Modo	TP	Errores	FP	FN	Exactitud (%)	Precisión (%)	Tiempo medio (ms)
Gafas	ESP32	17	3	0	0	85,00	100,00	883.35
	Servidor	20	0	0	0	100,00	100,00	774.20
Mascarilla	ESP32	2	9	0	0	18,18	100,00	841.82
	Servidor	1	19	0	0	5,00	100,00	785.45

La inclusión de accesorios redujo notablemente el rendimiento, especialmente en el ESP32-CAM. Con gafas, el nivel de acierto se redujo hasta un 85 %, mientras que el servidor mantuvo un 100 %.

Con mascarilla, los resultados fueron aún más drásticos: el ESP32-CAM solo alcanzó un 18 % de precisión, mostrando claras dificultades para reconocer rostros con la

boca cubierta. El servidor tampoco fue inmune a este problema, con una tasa de éxito del 5 %, aunque su descenso fue menos acusado.

Capacidad de usuarios

En lo relativo a la capacidad de almacenamiento de usuarios, el ESP32-CAM se encontró limitado por su arquitectura: al cargar todos los *embeddings* en la RAM, solo pudo manejar hasta 11 usuarios simultáneos.

El servidor, en cambio, no presentó esta restricción y logró manejar más de 50 usuarios sin degradación perceptible del rendimiento, aunque no se efectuaron *benchmarks* de carga exhaustivos. Por tanto, esta cifra debe entenderse como un resultado preliminar, y será necesario realizar evaluaciones específicas de escalabilidad en escenarios con un mayor número de sujetos y accesos concurrentes.

En definitiva, el ESP32-CAM mostró un comportamiento aceptable en entornos controlados, pero su rendimiento se redujo significativamente ante condiciones adversas como ángulos extremos, distancias superiores a 1 metro o el uso de mascarillas. El modo servidor, por su parte, demostró ser mucho más robusto y preciso en casi todas las pruebas, aunque con tiempos de respuesta más elevados en condiciones exigentes y una dependencia evidente de la calidad de la conexión de red.

6.7.5 Evaluación global

Los resultados confirman que el sistema cumple los requisitos funcionales: captura estable, reconocimiento fiable en condiciones normales, gestión de usuarios y trazabilidad de accesos.

En términos comparativos:

- **Modo local (ESP32-CAM):** tiempos muy estables (≈850 ms en condiciones normales), autónomo y suficiente para entornos controlados y con pocos usuarios. Presenta limitaciones marcadas ante ángulos pronunciados, distancias >1 m y accesorios (especialmente mascarilla). Capacidad simultánea limitada por RAM (≈11 usuarios).
- **Modo remoto (Servidor):** más robusto ante ángulos, múltiples rostros y accesorios; mantiene exactitud alta y, en las pruebas realizadas, pudo manejar más de 50 usuarios, aunque no se evaluó su límite real de escalabilidad. A cambio, su latencia es más variable (dependiente de red), con picos >2000 ms

aunque el promedio sea competitivo (≈878 ms) y, en algunos casos, incluso menor que el modo local.

Si bien en varios escenarios se alcanzó un 100 % de exactitud y precisión, este resultado refleja principalmente la baja variabilidad del conjunto de pruebas. La generalización de la eficacia del sistema requiere ampliar el número de usuarios y escenarios evaluados, cuestión que se contempla como trabajo futuro.

Como conclusión, tenemos que ambos modos son complementarios: el ESP32-CAM es preferible cuando priman autonomía, coste y simplicidad; el servidor, cuando se necesita precisión, robustez y escalabilidad. La elección depende del escenario de despliegue.

6.8. Estudio de coste y planificación

6.8.1. Coste del desarrollo

El sistema se ha concebido como un prototipo de bajo coste, empleando un único módulo central y software libre.

Tabla 6.7: Coste aproximado del prototipo.

Concepto	Coste aproximado
Módulo ESP32-CAM (OV2640)	10 €
Cables y adaptadores	3 €
Total hardware	≈ 13 €
Software (Arduino IDE, FastAPI, Flutter, SQLite, InsightFace)	0 €

El coste total del prototipo es inferior a 15 €, muy por debajo de los precios de sistemas comerciales (300–1000 €), lo que refuerza la viabilidad y accesibilidad de la solución propuesta.

No obstante, además del hardware, deben considerarse otros factores asociados al desarrollo:

- **Coste de desarrollo humano:** según el cronograma de trabajo (≈4 meses), se estima una dedicación total de entre 250 y 300 horas. Valoradas con el sueldo base promedio de un programador júnior en España (≈10–11 €/h, equivalente a 20 000 €/año), supondría un rango de 2500–3300 €.

- **Infraestructura empleada:** PC personal, conexión de red y consumo eléctrico. Aunque no supusieron un gasto adicional en este contexto, en un proyecto profesional deberían imputarse como costes de infraestructura.
- **Posible despliegue futuro:** en un escenario real sería necesario un servidor en la nube para el procesamiento facial. El coste de una instancia básica oscila entre 10–30 €/mes, según el proveedor y la concurrencia prevista.

En conclusión, el coste material del prototipo es mínimo, pero el desarrollo humano y la infraestructura representan una parte significativa en caso de extrapolar el proyecto a un entorno real.

6.8.2. Planificación del desarrollo

El proyecto se desarrolló a lo largo de 4 meses, distribuidos en fases secuenciales con cierta superposición entre ellas:

Tabla 6.8: Planificación del desarrollo del proyecto.

Fase	Duración estimada	Actividades principales
Investigación preliminar	3 semanas	Estudio de biometría, antecedentes y tecnologías disponibles
Diseño del sistema	3 semanas	Definición de arquitectura, modelos de datos y especificación de requisitos
Implementación del firmware	4 semanas	Programación del ESP32-CAM (captura de imágenes, detección local, comunicación).
Desarrollo del backend	4 semanas	Implementación en FastAPI, integración con InsightFace y SQLite.
Desarrollo de la app móvil	3 semanas	Creación de interfaz Flutter para gestión de usuarios y visualización de registros.
Pruebas y validación	3 semanas	Pruebas funcionales, de rendimiento y de robustez.
Redacción de la memoria	2 semanas	Documentación del proyecto y preparación de la defensa.

De esta manera, el trabajo se completó en un plazo de 4 meses, lo que confirma que la propuesta es viable dentro del calendario académico habitual y con una dedicación sostenida.

Capítulo 7

Conclusiones

7.1. Conclusiones generales

El presente trabajo ha mostrado que es posible llevar a cabo un sistema de control de acceso basado en reconocimiento facial utilizando un hardware muy asequible, como el ESP32-CAM, y apoyándose en herramientas de software libre como FastAPI, InsightFace, SQLite y Flutter.

Se han cumplido los objetivos principales: diseñar un sistema híbrido con dos modos de funcionamiento (procesamiento local en el ESP32-CAM y remoto en un servidor), desarrollar el firmware y los componentes de software necesarios, y comprobar su rendimiento en distintos escenarios.

Ahora bien, los resultados deben tomarse con cierta cautela. El sistema ha funcionado de manera estable en pruebas controladas, pero hay varios factores que limitan la solidez de las conclusiones:

- La exactitud cercana al 100 % se explica en gran medida por la baja diversidad de usuarios y de situaciones de prueba, por lo que no se puede asegurar que se mantenga en contextos reales más variados.
- Las métricas empleadas (exactitud y precisión) permiten validar la funcionalidad, pero no describen en detalle el comportamiento en casos de error. Métricas más completas como *recall* o F1-score aportarían una visión más rica.
- El alcance de las pruebas también estuvo condicionado por el hardware (ESP32-CAM con cámara OV2640) y por la falta de experimentos prolongados, de modo que no es posible extrapolar los resultados a sistemas con mayor carga

o sensores diferentes.

- En el caso del modo servidor, aunque mostró mayor robustez que el dispositivo, no se hicieron pruebas con un número elevado de usuarios concurrentes, así que las conclusiones sobre su escalabilidad son solo preliminares.

En definitiva, el proyecto confirma que es viable plantear una solución híbrida y de bajo coste para el control de accesos mediante reconocimiento facial. Sin embargo, lo conseguido debe verse más como un punto de partida experimental que como una validación definitiva de su uso en entornos productivos reales.

7.2. Limitaciones y amenazas a la validez

Aunque el sistema ha demostrado un buen funcionamiento en las pruebas realizadas, es importante señalar varios aspectos que limitan la validez general de los resultados:

- **Condiciones de red:** en el modo servidor la latencia depende directamente de la calidad del Wi-Fi y de la carga de trabajo del equipo. Los picos de respuesta observados en las pruebas podrían no repetirse de la misma manera en otras redes o entornos de despliegue.
- **Diversidad de la muestra:** se hicieron muchos intentos de reconocimiento, pero con un grupo reducido de usuarios y condiciones. No se evaluaron variaciones amplias de edad, tono de piel, presencia de vello facial u otros factores que afectan al rendimiento en escenarios reales.
- **Medición de tiempos:** los tiempos registrados incluyen tanto la detección como el reconocimiento, pero no reflejan costes adicionales como el renderizado de la interfaz o el streaming continuo, que sí influyen en un uso práctico.
- **Hardware utilizado:** todo el sistema se probó únicamente con el ESP32-CAM y la cámara OV2640. El rendimiento podría variar si se usaran otros modelos de cámara, diferentes lentes o versiones de firmware.
- **Anti-spoofing:** no se realizaron pruebas contra intentos de suplantación mediante fotos, vídeos o máscaras, por lo que la seguridad frente a estos ataques sigue siendo una incógnita.
- **Aplicación móvil:** la app desarrollada en Flutter solo se probó en emulador (Android Studio). No se midieron latencias de notificación ni la experiencia real de uso en dispositivos físicos.

- **Iluminación:** se hicieron pruebas con diferentes condiciones de luz, pero sin instrumentación (por ejemplo, un luxómetro). Esto dificulta la comparación con trabajos que sí aportan medidas objetivas de intensidad lumínica.
- **Pruebas prolongadas:** no se realizaron ensayos de larga duración para detectar posibles fugas de memoria o fallos de estabilidad tras varias horas de funcionamiento.
- **Escalabilidad:** en modo servidor se comprobó que era capaz de manejar más de 50 usuarios, pero no se hicieron benchmarks específicos de carga, así que no puede asegurarse su comportamiento en escenarios con muchos accesos concurrentes.

Estas limitaciones no restan valor al prototipo, pero sí marcan claramente el alcance de los resultados: se trata de una validación en un entorno controlado, no de una certificación de uso en condiciones productivas.

7.3. Trabajos futuros

El sistema desarrollado constituye una base sólida sobre la que se pueden plantear mejoras y ampliaciones en el futuro. Algunas líneas de trabajo que se consideran especialmente relevantes son:

- **Seguridad:** incorporar cifrado en las comunicaciones entre el ESP32 y el servidor, así como mecanismos de *anti-spoofing* para evitar accesos mediante fotos, vídeos o máscaras.
- **Ampliación de pruebas:** repetir los experimentos con un mayor número de usuarios y en condiciones más variadas (diferentes edades, rasgos, iluminación o entornos de uso), lo que permitiría obtener métricas más realistas.
- **Métricas avanzadas:** además de la exactitud y la precisión, incluir indicadores como *recall* y F1-score que permitan analizar mejor los casos de error y el equilibrio entre falsos positivos y falsos negativos.
- **Escalabilidad:** realizar benchmarks específicos para evaluar hasta qué punto el servidor puede gestionar usuarios concurrentes de forma estable, y en qué condiciones empieza a degradar su rendimiento.
- **Aplicación móvil:** probar la app en dispositivos reales, optimizar la interfaz y mejorar la experiencia de uso con notificaciones fiables y un diseño adaptado a un despliegue práctico.

- **Integración de hardware adicional:** conectar el sistema con cerraduras electrónicas u otros mecanismos físicos de control, para acercar el prototipo a un escenario de aplicación real.
- **Alternativas de hardware:** explorar módulos con mayor memoria o cámaras con mejor sensibilidad en baja luz, con el fin de superar algunas de las limitaciones detectadas en el ESP32-CAM.
- **Estabilidad a largo plazo:** llevar a cabo pruebas continuas de varias horas o días que permitan detectar problemas de degradación, fugas de memoria o fallos en la comunicación de red.

En conjunto, estas líneas de trabajo abren el camino hacia una versión más madura y completa del sistema, que combine el bajo coste y la simplicidad del prototipo con las exigencias de seguridad, fiabilidad y escalabilidad propias de un entorno real de producción.

Bibliografía

[1] RobotZero1. Esp32-cam access control, 2019. URL `https://github.com/robotzero1/esp32cam-access-control`. GitHub repository.

[2] Rui Santos. Esp32-cam camera web server, 2020. URL `https://github.com/RuiSantosdotme/arduino-esp32-CameraWebServer`. GitHub repository.

[3] X. Lv, M. Su, and Z. Wang. Application of face recognition method under deep learning algorithm in embedded systems. *Microprocessors and Microsystems*, 80:103364, 2021. doi: 10.1016/j.micpro.2021.103364. URL `https://www.sciencedirect.com/science/article/pii/S0141933121002064`.

[4] Z. Chen, J. Chen, G. Ding, and H. Huang. A lightweight cnn-based algorithm and implementation on embedded system for real-time face recognition. *Multimedia Systems*, 2023. doi: 10.1007/s00530-022-00973-z. URL `https://link.springer.com/article/10.1007/s00530-022-00973-z`.

[5] J.-J. Wu. Efficient facial landmark detection for embedded systems. *arXiv preprint*, 2024. URL `https://arxiv.org/abs/2407.10228`.

[6] N. A. Andriyanov and V. E. Dementiev. Optimization of face recognition systems for implementation in embedded systems. *Pattern Recognition and Image Analysis*, 34(4):765–776, 2024. doi: 10.1134/S1054661824701311. URL `https://link.springer.com/article/10.1134/S1054661824701311`.

[7] T. Abderraouf, A. A. Wassim, and S. Larabi. An embedded intelligent system for attendance monitoring. *arXiv preprint*, 2024. URL `https://arxiv.org/abs/2406.13694`.

[8] B.-T. Nguyen-Tat, M.-Q. Bui, and V. M. Ngo. Automating attendance management in human resources: A design science approach using computer vision and facial recognition. *International Journal of Information Management Data Insights*, 2024. URL `https://arxiv.org/abs/2109.03398`.

[9] S. Serengil and A. Ozpinar. Cipherface: A fully homomorphic encryption-driven framework for secure cloud-based facial recognition. *arXiv preprint*, 2025. URL https://arxiv.org/abs/2502.18514.

[10] ZKTeco. Speedface v5l – sistema de reconocimiento facial, 2023. URL https://zktecotiendaoficial.com/product/speedface-v5l-p-wifi.

[11] Amazon. Terminal de asistencia speedface v5l, 2023. URL https://www.amazon.es/asistencia-reconocimiento-SpeedFace-V5L-Seguimiento-Speedface-V5L/dp/B0D5CPTVF3.

[12] Suprema. Facelite fl-db terminal, 2023. URL https://www.logiscenter.com/suprema-facelite.

[13] Raspberry Pi Foundation. ¿qué es raspberry pi?, 2023. URL https://raspberrypi.cl/que-es-raspberry/.

[14] Programar Fácil. Introducción al esp32, 2023. URL https://programarfacil.com/esp8266/esp32/.

[15] Programar Fácil. Esp32-cam: qué es, características y cómo reprogramarlo, 2023. URL https://programarfacil.com/esp32/esp32-cam/.

[16] OpenWebinars. Introducción a opencv y visión por computadora, 2023. URL https://openwebinars.net/blog/opencv-introduccion-y-su-rol-en-la-vision-por-computadora/.

[17] DeepInsight. Insightface – deep face analysis toolkit, 2023. URL https://github.com/deepinsight/insightface.

[18] DeepInsight. Insightface: Python package, 2025. URL https://github.com/deepinsight/insightface/tree/master/python-package.

[19] Arduino. Arduino ide, 2023. URL https://es.wikipedia.org/wiki/Arduino_IDE.

[20] Espressif. Esp-who: Face detection and recognition framework, 2023. URL https://github.com/espressif/esp-who.

[21] PlatformIO. Platformio – entorno de desarrollo, 2023. URL https://platformio.org/.

[22] Arsys. Qué es flask y cuáles son sus ventajas, 2023. URL https://www.arsys.es/blog/que-es-flask-python-y-cuales-son-sus-ventajas.

[23] AWS. ¿qué es django?, 2023. URL https://aws.amazon.com/es/what-is/django/.

[24] FastAPI. Fastapi documentation, 2023. URL https://fastapi.tiangolo.com.

[25] Meta Platforms, Inc. React native – create native apps for android and ios using react, 2019. URL https://reactnative.dev/.

[26] Flutter. Flutter framework – documentación oficial, 2023. URL https://esflutter.dev/.

[27] A. Allan. Face detection and recognition on the esp32, 2019. URL https://aallan.medium.com/face-detection-and-recognition-on-the-esp32-3b4b9a35c765. Medium article.

[28] SQLite Consortium. Sqlite, 2025. URL https://www.sqlite.org/.

[29] Ultralytics. Residual networks (resnet), 2023. URL https://www.ultralytics.com/es/glossary/residual-networks-resnet.

Anexo A

Repositorio del proyecto

El código fuente completo desarrollado en este Trabajo de Fin de Grado se encuentra disponible públicamente en GitHub. Este repositorio contiene tanto el firmware del ESP32-CAM como el servidor backend y la aplicación móvil.

- Repositorio en GitHub: **github.com/FranciscoOR-08/TFG**