UNIVERSIDAD NACIONAL PEDRO RUÍZ GALLO
FACULTAD DE INGENIERÍA CIVIL, DE SISTEMAS Y DE ARQUITECTURA
ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

TESIS

Dispositivo inteligente basado en reconocimiento facial para el registro de asistencia

PARA OBTENER EL TÍTULO PROFESIONAL DE:

Ingeniero(a) de sistemas

AUTORES

Bach. Ramirez Guevara Maikol

Bach. Rangel Quispe Maricielo del Carmen

ASESOR

Dr. Ing. Villegas Cubas Juan Elías

Lambayeque, 2026

UNIVERSIDAD NACIONAL PEDRO RUÍZ GALLO

FACULTAD DE INGENIERÍA CIVIL, DE SISTEMAS Y DE ARQUITECTURA

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

TESIS

Dispositivo inteligente basado en reconocimiento facial para el registro de asistencia

PARA OPTAR EL TÍTULO PROFESIONAL DE INGENIERIA DE SISTEMAS

APROBADO POR EL SIGUIENTE JURADO:

Msc. Ing. Omar Wilton Saavedra Salazar

Presidente

Msc. Ing. Roberto Carlos Arteaga Lora

Secretario

Msc. Ing. Oscar Efraín Capuñay Uceda

Vocal

Dr. Ing. Juan Elias Villegas Cubas

Asesor

UNIVERSIDAD NACIONAL PEDRO RUÍZ GALLO
FACULTAD DE INGENIERÍA CIVIL,DE SISTEMAS Y DE ARQUITECTURA
ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

TESIS

Dispositivo inteligente basado en reconocimiento facial para el registro de asistencia

PARA OPTAR EL TÍTULO PROFESIONAL DE INGENIERIA DE SISTEMAS

ELABORADO POR:

Bach. Maikol Ramirez Guevara

Autor

Bach. Maricielo del Carmen Rangel Quispe

Autor

UNIVERSIDAD NACIONAL PEDRO RUIZ GALLO
FACULTAD DE INGENIERÍA CIVIL DE SISTEMAS Y DE ARQUITECTURA
UNIDAD DE INVESTIGACIÓN

ACTA DE SUSTENTACIÓN N°009-2026-UI-FICSA

Siendo las 12:30 pm horas del día 12 de junio del 2026, se reunieron de manera presencial los miembros de jurado de la tesis titulada: **"DISPOSITIVO INTELIGENTE BASADO EN ECONOCIMIENTO FACIAL PARA EL REGISTRO DE ASISTENCIA", con código N° IS_V_2025_015,** designado por **Resolución Decanal Virtual N° 254-2025-UNPRG-FICSA** con la finalidad de Evaluar y Calificar la sustentación de la tesis antes mencionada, conformado por los siguientes docentes:

MSC. ING. OMAR WILTON SAAVEDRA SALAZAR	**PRESIDENTE**
MSC. ING. ROBERTO CARLOS ARTEAGA LORA	**SECRETARIO**
MSC. ING. OSCAR EFRAIN CAPUÑAY UCEDA	**VOCAL**

Asesorado por DR. ING. JUAN ELIAS VILLEGAS CUBAS.

El acto de sustentación fue autorizado por **OFICIO VIRTUAL N° 134-2026-UIFICSA**, la tesis fue presentada y sustentada por los Bachilleres: **MARICIELO DEL CARMEN RANGEL QUISPE y MAIKOL RAMIREZ GUEVARA,** tuvo una duración de 55 minutos. Después de la sustentación, y absueltas las preguntas y observaciones de los miembros del jurado; se procedió a la calificación respectiva:

	NUMERO	LETRAS	CALIFICATIVO
MARICIELO DEL CARMEN RANGEL QUISPE	17	DIECISIETE	BUENO
MAIKOL RAMIREZ GUEVARA	17	DIECISIETE	BUENO

Por lo que quedan **APTOS** para obtener el Título Profesional de **INGENIERO (A) DE SISTEMAS** de acuerdo con la Ley Universitaria 30220 y la normatividad vigente de la Facultad de Ingeniería Civil De Sistemas y de Arquitectura de la Universidad Nacional Pedro Ruiz Gallo.

Siendo las 1:25PM Se dio por concluido el presente acto académico, dándose conformidad al presente acto, con la firma de los miembros del jurado.

MSC. ING. OMAR WILTON SAAVEDRA SALAZAR
PRESIDENTE

MSC. ING. ROBERTO CARLOS ARTEAGA LORA
SECRETARIO

MSC. ING. OSCAR EFRAIN CAPUÑAY UCEDA
VOCAL

DR. ING. JUAN ELIAS VILLEGAS CUBAS
ASESOR

JHFF/bcm

UNIVERSIDAD NACIONAL PEDRO RUIZ GALLO
FACULTAD DE INGENIERÍA CIVIL, DE SISTEMAS Y DE ARQUITECTURA
UNIDAD DE INVESTIGACIÓN

CONSTANCIA DE VERIFICACIÓN DE ORIGINALIDAD

Yo, Dr. Ing. Juan Elias Villegas Cubas, **asesor de tesis de los bachilleres:**

Maikol Ramirez Guevara y

Maricielo del Carmen Rangel Quispe

TITULADA:

"DISPOSITIVO INTELIGENTE BASADO EN RECONOCIMIENTO FACIAL PARA EL REGISTRO DE ASISTENCIA"

Luego de la revisión exhaustiva del documento constato que la misma tiene un índice de similitud de 11% verificable en el reporte de similitud del programa TURNITIN.

El suscrito analizó dicho reporte y concluyó que cada una de las coincidencias detectadas NO CONSTITUYEN PLAGIO. A mi leal saber y entender la tesis cumple con todas las normas para el uso de citas y referencias establecidas por la Universidad Nacional Pedro Ruiz Gallo.

Se expide la presente según lo dispuesto en la Resolución Nº 659-2020-R, de fecha 8 de setiembre de 2020 formativa para la obtención de Grados y Títulos de la UNPRG:

Lambayeque, 31 de marzo del 2026

ATENTAMENTE,

Dr. Ing. Juan Elias Villegas Cubas
DNI. 80103991
Asesor

Se adjunta:
Recibo digital de Turnitin
Revisión de informe en Turnitin

Dispositivo Inteligente basado en reconocimiento facial para el registro de asistencia

INFORME DE ORIGINALIDAD

11%	10%	2%	4%
INDICE DE SIMILITUD	FUENTES DE INTERNET	PUBLICACIONES	TRABAJOS DEL ESTUDIANTE

Dr. Ing. Villegas Cubas
Juan Elias
DNI. 80103991
Asesor

FUENTES PRIMARIAS

1	repositorio.unprg.edu.pe Fuente de Internet	2%
2	www.coursehero.com Fuente de Internet	1%
3	Submitted to Universidad Nacional Pedro Ruiz Gallo Trabajo del estudiante	<1%
4	hdl.handle.net Fuente de Internet	<1%
5	Submitted to Instituto Superior de Artes, Ciencias y Comunicación IACC Trabajo del estudiante	<1%
6	1library.co Fuente de Internet	<1%
7	repositorio.uss.edu.pe Fuente de Internet	<1%
8	repositorio.upse.edu.ec Fuente de Internet	<1%
9	www.slideshare.net Fuente de Internet	<1%
10	Submitted to Universidad Tecnologica del Peru Trabajo del estudiante	<1%
	www.mdpi.com	

11	Fuente de Internet	<1 %
12	laccei.org Fuente de Internet	<1 %
13	repositorio.ucv.edu.pe Fuente de Internet	<1 %
14	apirepositorio.unu.edu.pe Fuente de Internet	<1 %
15	dspace.utpl.edu.ec Fuente de Internet	<1 %
16	core.ac.uk Fuente de Internet	<1 %
17	es.scribd.com Fuente de Internet	<1 %
18	repositorio.unica.edu.pe Fuente de Internet	<1 %
19	repositorio.unc.edu.pe Fuente de Internet	<1 %
20	Submitted to Institución Universitaria Digital de Antioquia Trabajo del estudiante	<1 %
21	Submitted to Universidad Estatal Amazonica- Trabajo del estudiante	<1 %
22	Submitted to AULA VIRTUAL Trabajo del estudiante	<1 %
23	Submitted to Universidad TecMilenio Trabajo del estudiante	<1 %
24	empiezoinformatica.wordpress.com Fuente de Internet	<1 %
25	sembrandoelmar.cl Fuente de Internet	<1 %

Dr. Ing. Villegas Cubas
Juan Elias
DNI. 80103991
Asesor

26	Submitted to uncedu Trabajo del estudiante	<1 %
27	Submitted to Universidad Internacional de la Rioja Trabajo del estudiante	<1 %
28	postgrados.cusam.edu.gt Fuente de Internet	<1 %
29	Submitted to uamerica Trabajo del estudiante	<1 %
30	Submitted to Universidad APEC Trabajo del estudiante	<1 %
31	Submitted to Universidad de León Trabajo del estudiante	<1 %
32	Submitted to Universidad Nacional de Cañete Trabajo del estudiante	<1 %
33	Submitted to Universidad Nacional de Trujillo Trabajo del estudiante	<1 %
34	Submitted to Universidad de Alcalá Trabajo del estudiante	<1 %
35	Submitted to Universidad Nacional de Tumbes Trabajo del estudiante	<1 %
36	digital.csic.es Fuente de Internet	<1 %
37	doaj.org Fuente de Internet	<1 %
38	docplayer.es Fuente de Internet	<1 %
39	view.genial.ly Fuente de Internet	<1 %

Dr. Ing. Villegas Cubas
Juan Elias
DNI. 80103991
Asesor

40	Alanoca Gutierrez, Isabel. "Influencia del sistema de administración financiera SIAF -SP en la presentación de los estados financieros de las municipalidades provinciales de la región de Puno - 2022", Universidad Nacional del Altiplano de Puno (Peru) Publicación	<1 %
41	repositorio.uancv.edu.pe Fuente de Internet	<1 %
42	www.sudoc.fr Fuente de Internet	<1 %
43	Submitted to Universidad de Guayaquil Trabajo del estudiante	<1 %
44	atetic.ulpgc.es Fuente de Internet	<1 %
45	gredos.usal.es Fuente de Internet	<1 %
46	repository.udistrital.edu.co Fuente de Internet	<1 %
47	riujap.ujap.edu.ve Fuente de Internet	<1 %
48	upc.aws.openrepository.com Fuente de Internet	<1 %
49	www.domoprac.com Fuente de Internet	<1 %
50	www.investigacion360.com Fuente de Internet	<1 %
51	www.notebookcheck.org Fuente de Internet	<1 %
52	www.unicef.org Fuente de Internet	<1 %

Dr. Ing. Villegas Cubas
Juan Elias
DNI. 80103991
Asesor

turnitin

Recibo digital

Este recibo confirma quesu trabajo ha sido recibido por Turnitin. A continuación podrá ver la información del recibo con respecto a su entrega.

La primera página de tus entregas se muestra abajo.

Autor de la entrega:	Ramirez Guevara Maikol Rangel Quispe Maricielo
Título del ejercicio:	Quick Submit
Título de la entrega:	Dispositivo Inteligente basado en reconocimiento facial para e…
Nombre del archivo:	Entregable.docx
Tamaño del archivo:	8.93M
Total páginas:	87
Total de palabras:	14,502
Total de caracteres:	83,979
Fecha de entrega:	31-mar-2026 05:30p. m. (UTC-0500)
Identificador de la entrega:	2919184310

UNIVERSIDAD NACIONAL PEDRO RUÍZ GALLO
FACULTAD DE INGENIERÍA CIVIL, DE SISTEMAS Y DE ARQUITECTURA
ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

TESIS

Dispositivo inteligente basado en reconocimiento facial para el registro de asistencia

AUTORES

Bach. Ramirez Guevara Maikol

Bach. Rangel Quispe Maricielo del Carmen

ASESOR

Dr. Ing. Villegas Cubas Juan Elias

Lambayeque, 2026

Dr. Ing. Villegas Cubas
Juan Elias
DNI. 80103991
Asesor

DEDICATORIA

Dedicamos este trabajo, fruto de nuestro esfuerzo, constancia y dedicación, a nuestras familias, por su apoyo y su motivación constante a lo largo de este camino.

A nuestros padres, quienes con su ejemplo nos enseñaron el valor del compromiso y la perseverancia, siendo el pilar fundamental en cada uno de nuestros logros.

Asimismo, a las personas que nos otorgaron su apoyo y ánimo durante el desarrollo de la investigación.

Finalmente, nos dedicamos este logro a nosotros mismos, por no rendirnos ante las dificultades y demostrar que con trabajo en equipo y determinación es posible alcanzar nuestras metas.

AGRADECIMIENTO

Agradecemos a Dios por acompañarnos con su sabiduría, darnos la fuerza necesaria en los momentos difíciles y permitirnos alcanzar con éxito esta etapa tan significativa de nuestra formación profesional.

A nuestros padres, por su amor único, sacrificio y apoyo constante, siendo el pilar fundamental en cada etapa de nuestra formación. Su confianza y aliento han sido esenciales para no rendirnos ante las dificultades.

A nuestras familias, por su comprensión y motivación permanente a lo largo de este proceso.

A nuestro asesor, por su guía, paciencia y dedicación durante el desarrollo de esta investigación.

Finalmente, a todas las personas que, de una u otra manera, contribuyeron a la realización de este trabajo.

Los autores.

RESUMEN

El presente proyecto de investigación tecnológica tiene como objetivo implementar un dispositivo inteligente basado en reconocimiento facial para el registro automatizado de asistencia. La problemática identificada se centra en las deficiencias de los sistemas manuales de control de asistencia, los cuales generan pérdidas de tiempo, errores frecuentes y riesgos de suplantación de identidad en diversas instituciones educativas y empresariales. La investigación es de tipo aplicada, de enfoque cuantitativo, y con diseño experimental; así mismo la metodología se estructura en cinco etapas: la identificación y selección de herramientas de reconocimiento facial, donde se evaluaron mobilefaceNet, edgeface-s(gamma_05) y edgeface-xs(gamma_06); el desarrollo del software de reconocimiento facial, implementado mediante Python y estructurado en tres módulos (base de datos SQLite, backend para el procesamiento de reconocimiento, y frontend para la interfaz de usuario); el diseño de la arquitectura del dispositivo inteligente integrando hardware (Raspberry Pi 4B, cámara, pantalla LCD) y software; construcción del dispositivo inteligente; y la evaluación del rendimiento de dicho dispositivo. Los resultados muestran que la combinación de estas tecnologías permite la detección y reconocimiento facial en tiempo real, con un procesamiento eficiente de las imágenes capturadas, logrando una exactitud de 99.00%, una precisión de 99.01%, una sensibilidad de 99.01%, y un 98.99% de especificidad. Se concluye que el dispositivo inteligente es eficiente en el registro de control de asistencia.

Palabras clave: Reconocimiento facial, dispositivo inteligente, registro de asistencia, Raspberry Pi, visión artificial, mobilefaceNet, sistema embebido

ABSTRACT

This technological research project aims to implement an intelligent device based on facial recognition for automated attendance registration. The identified problem focuses on the deficiencies of manual attendance control systems, which lead to time loss, frequent errors, and risks of identity impersonation in various educational and business institutions. The research is applied in nature, with a quantitative approach and an experimental design; likewise, the methodology is structured into five stages: the identification and selection of facial recognition tools, where MobileFaceNet, EdgeFace-S (gamma_05), and EdgeFace-XS (gamma_06) were evaluated; the development of the facial recognition software, implemented in Python and structured into three modules (SQLite database, backend for recognition processing, and frontend for the user interface); the design of the intelligent device architecture integrating hardware (Raspberry Pi 4B, camera, LCD screen) and software; the construction of the intelligent device; and the performance evaluation of the device. The results show that the combination of these technologies enables real-time facial detection and recognition, with efficient processing of captured images, achieving an accuracy of 99.00%, precision of 99.01%, sensitivity of 99.01%, and specificity of 98.99%. It is concluded that the intelligent device is efficient for attendance control registration.

Keywords: Facial recognition, smart device, attendance recording, Raspberry Pi, computer vision, mobilefaceNet, embedded system

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ANEXOS

INTRODUCCIÓN

En los últimos años, el uso de tecnologías inteligentes para optimizar procesos administrativos y académicos ha cobrado gran relevancia. Entre estos procesos, el registro de asistencia se mantiene como una actividad crucial para la gestión institucional, ya que permite controlar la participación, monitorear el cumplimiento de responsabilidades y garantizar una adecuada organización interna.

Sin embargo, en muchas instituciones este procedimiento continúa realizándose de manera manual, lo que genera demoras, inconsistencias y dificultades para mantener datos precisos y actualizados. Esta situación se vuelve más evidente en contextos donde existe un alto flujo de estudiantes, docentes o personal, incrementando la probabilidad de errores y afectando la eficiencia de los procesos. En este escenario, surge la necesidad de explorar alternativas tecnológicas que permitan modernizar y automatizar los sistemas de control de asistencia, de modo que las instituciones puedan disponer de información confiable y gestionar sus actividades con mayor precisión y agilidad.

Por otro lado, la presente investigación se estructura de manera sistemática y coherente, siguiendo un esquema que garantiza la rigurosidad científica y la adecuada organización del contenido. En primer lugar, la Introducción presenta el contexto general del estudio, destacando la relevancia del tema, la justificación de la investigación y una visión global del problema abordado.

Seguidamente, el Capítulo II: Planteamiento de la investigación desarrolla los elementos fundamentales que orientan el estudio. En este apartado se incluye la síntesis de la situación problemática, donde se describe y contextualiza el problema; la formulación del problema de investigación, que delimita de manera precisa la interrogante principal; y finalmente los objetivos, tanto general como específicos, que definen los alcances de la investigación.

El Capítulo III: Diseño teórico reúne el sustento conceptual del estudio. En esta sección se presentan los antecedentes, que permiten conocer investigaciones previas relacionadas; las bases teóricas, que contienen los enfoques, modelos y teorías que fundamentan el estudio; y las bases conceptuales, donde se definen los términos clave y las variables de investigación.

Posteriormente, el Capítulo IV: Diseño metodológico detalla la estrategia seguida para el desarrollo del estudio. Incluye el diseño de contrastación y procedimiento, que describe el tipo, enfoque y diseño de investigación, así como las etapas del proceso; la población y muestra, donde se especifican los elementos de estudio; y las técnicas, instrumentos, equipos y materiales, que explican los medios utilizados para la recolección y análisis de datos.

El Capítulo V: Resultados presenta de manera objetiva los hallazgos obtenidos, generalmente apoyados en tablas, gráficos o figuras, sin interpretación extensa.

En el Capítulo VI: Discusión de resultados, se interpretan los hallazgos, contrastándolos con los antecedentes, estableciendo comparaciones, explicaciones y posibles implicancias.

El Capítulo VII: Conclusiones sintetiza los principales resultados en función de los objetivos planteados, evidenciando el cumplimiento de estos. A continuación, se formulan las recomendaciones, orientadas a la aplicación práctica de los resultados o a futuras líneas de investigación.

Finalmente, se incluyen las referencias, que consignan las fuentes bibliográficas utilizadas bajo normas académicas, y los anexos, donde se incorporan documentos complementarios que respaldan la investigación, como instrumentos, evidencias o información adicional relevante.

2. PLANTEAMIENTO DE LA INVESTIGACIÓN

2.1 Síntesis de la situación problemática

Actualmente a nivel internacional, existen diferentes métodos para registrar la asistencia en las aulas de cada institución educativa, siendo el método manual el que más predomina a pesar de los avances tecnológicos; de esta manera, algunos recurren a formularios manuales, mientras que otros no consideran relevante llevar un control sistemático de la asistencia (Mondal y Mondal, 2023). El registro manual de asistencia consume tiempo de la clase, ya que los docentes deben dedicar parte de la sesión a este proceso, cuyo esfuerzo aumenta en función de la cantidad de estudiantes matriculados, lo cual genera un problema en las diversas instituciones (Ishaq y Bibi, 2023).

En el Perú, a nivel nacional, el control de asistencia en diversas instituciones públicas y privadas es un proceso poco eficiente debido a su carácter manual. En el caso del Programa Nacional de Apoyo Directo a los Más Pobres (JUNTOS), se identificó que los registros tradicionales de la asistencia del personal realizada en papel generan retrasos administrativos, errores frecuentes y posibilidades de pérdida de información, lo que limita la adecuada gestión de los recursos y dificulta alcanzar los objetivos organizacionales (Anyaypoma y Hoyos , 2021). Una situación similar ocurre en el ámbito empresarial, donde compañías como Global Sales Solutions Line Sucursal Perú, enfrentaban dificultades derivadas de la dependencia de hojas de asistencias firmadas por los trabajadores, lo que no solo ocasionaba demoras, sino también problemas de confiabilidad de los datos y riesgos de suplantación de identidad (Espino, 2019). Estos ejemplos evidencian que, a nivel nacional, la falta de sistemas automatizados de control de asistencia afecta directamente la productividad, la transparencia y la optimización de recursos en distintos sectores.

En la región Lambayeque, persisten limitaciones en la adopción de tecnologías orientadas a optimizar los procesos administrativos y académicos, especialmente en el registro de asistencia de los estudiantes. Un estudio realizado en 2023 evidenció que este proceso aún se desarrolla, en muchos casos, de forma manual o con escasa automatización, lo que genera demoras, posibles errores y baja eficiencia en la gestión de los registros (De La Cruz, 2023). De igual modo, una investigación en la UGEL de Lambayeque identificó deficiencias en la calidad de la información relacionada con la asistencia, atribuibles a la limitada utilización de herramientas tecnológicas y a la escasa digitalización de los datos, lo que dificulta un control oportuno y confiable de la presencia estudiantil (Lau, 2022). De forma análogo, en el Grupo de investigación en Ciencia de Datos, Inteligencia de Datos y Ciberseguridad, de la región, se ha observado la carencia de un dispositivo inteligente para el registro de asistencias; pues actualmente este control se realiza manualmente, lo que consume tiempo, puede generar omisiones y dificulta la verificación de la información. Por ello, resulta necesario implementar un sistema automatizado para el registro de asistencias y garantice mayor precisión y confiabilidad en los datos.

En conjunto, el análisis en los tres niveles: internacional, nacional y local, permite evidenciar que el control de asistencia sigue siendo un proceso crítico que, al mantenerse bajo métodos manuales, expone a las organizaciones a riesgos de ineficiencia, errores y pérdida de información. Por lo mencionado, la adopción de tecnologías de automatización en el registro de asistencia, se presenta como una alternativa indispensable para garantizar procesos más confiables, transparentes y acordes con las demandas actuales, especialmente para docentes, tesistas, practicantes e investigadores en formación, quienes requieren sistemas que respalden sus actividades con datos precisos y verificables.

2.2 Formulación del problema de investigación

¿Qué tan eficiente es un dispositivo inteligente basado en reconocimiento facial para el registro de asistencia?

2.3 Hipótesis

Este proyecto se clasifica dentro de la categoría de investigación tecnológica, ya que su principal objetivo es el diseño, desarrollo e implementación de un sistema innovador que integra tecnologías avanzadas de reconocimiento facial y sistemas embebidos para el registro automatizado de asistencia. A continuación, se detallan los aspectos clave que sustentan esta clasificación:

- **Fundamento tecnológico:** El proyecto se basa en el uso de técnicas avanzadas de reconocimiento facial, apoyadas en algoritmos de inteligencia artificial, como las redes neuronales convolucionales (CNN), y modelos pre entrenados. Estas herramientas permiten la identificación precisa de individuos a partir de imágenes o video en tiempo real, lo que se complementa con el uso de hardware embebido, específicamente una Raspberry Pi 4B, que actúa como la plataforma central del sistema.
- **Aporte innovador:** La propuesta representa una solución tecnológica novedosa en el ámbito del control de asistencia, al reemplazar métodos tradicionales (como listas manuales o tarjetas de identificación) con un sistema automatizado y no invasivo.
- **Desarrollo y validación práctica:** El proyecto incluye la construcción y evaluación de un prototipo funcional, diseñado para operar en entornos reales, como instituciones educativas o empresas. Esta fase de validación permite demostrar la viabilidad y efectividad del sistema, asegurando que cumpla con los requisitos técnicos y las expectativas de los usuarios finales.

Dado que este trabajo se centra en el desarrollo de una solución tecnológica y no en la comprobación de relaciones causa-efecto, no se plantea una hipótesis formal. En su lugar, el enfoque se dirige hacia la consecución de objetivos, orientados a la creación y validación de un dispositivo que resuelva un problema en el ámbito del registro de asistencia.

2.4 Objetivos

2.4.1 *Objetivo general*

Implementar un dispositivo inteligente basado en reconocimiento facial para el registro de asistencia.

2.4.2 *Objetivos específicos*

- Identificar y seleccionar herramientas para el reconocimiento y detección de rostros.
- Desarrollar un software, incorporando herramientas de reconocimiento facial.
- Diseñar la arquitectura del dispositivo inteligente de registro de asistencia.
- Construir el dispositivo inteligente.
- Evaluar el rendimiento del dispositivo inteligente en el registro de asistencia.

3. DISEÑO TEÓRICO

3.1 Antecedentes

Ogla et al. (2022) tuvieron como propósito diseñar un método de reconocimiento facial combinando momentos con el histograma de patrones binarios locales. El estudio fue aplicado, de enfoque cuantitativo y con un diseño matexperimental. La población de estudio fueron imágenes del Laboratorio Olivetti,

utilizando una muestra de 400 fotografías correspondientes a 40 personas. Para la extracción de características se aplicaron algoritmos basados en momentos y LBP. Los hallazgos demostraron que la estrategia híbrida aumentó la precisión en un 3%, logrando un total de 98,78%, lo cual evidenció que la combinación de técnicas resulta más efectiva que el uso aislado.

Ullah et al. (2022) propusieron un sistema de reconocimiento facial en tiempo real apoyado en técnicas de aprendizaje profundo. El trabajo, de naturaleza aplicada y con un diseño experimental, empleó más de 40,000 imágenes obtenidas de cámaras de vigilancia, capturadas bajo distintas condiciones de luz y ángulos. Como instrumentos de análisis se aplicaron PCA, DT, RF, KNN y una CNN. El rendimiento alcanzado con la CNN fue de 95,7% usando un conjunto reducido de imágenes para entrenamiento y prueba. Se concluyó que las CNN permiten un reconocimiento robusto incluso en entornos de seguridad con grandes volúmenes de datos.

Al-Ghrairi (2022) buscó implementar un sistema ágil de reconocimiento facial que pudiera manejar expresiones diversas, variaciones de pose y rotaciones amplias. La investigación, de tipo aplicada, utilizó la base de datos FEI como población y dos subconjuntos de imágenes como muestra. El proceso incluyó etapas de preprocesamiento y detección de rostros mediante el algoritmo Viola-Jones. Los resultados mostraron un 96% de precisión tanto en rostros frontales como de perfil, con un bajo tiempo de procesamiento. Se concluyó que este modelo es confiable y eficiente para la detección en escenarios variados.

Ke et al. (2022) desarrollaron la técnica denominada LocalFace, orientada a extraer rasgos faciales locales. El trabajo fue aplicado, cuantitativo y experimental. La población correspondió a más de 85 mil sujetos, con una muestra de 5,8 millones de imágenes del dataset MS1MV2 refinado. Como herramientas se emplearon LocalFace

y MobileFaceNet V2. Los resultados evidenciaron un rendimiento del 88,27%, superior a métodos previos basados en oclusión aleatoria. Se concluyó que la extracción local de características es útil para mejorar la precisión en bases de datos de gran escala.

Archana et al. (2022) diseñaron un sistema de detección y mapeo facial para entornos de clases en línea. El estudio, de carácter aplicado y con diseño experimental, utilizó el dataset ORL16, compuesto por 400 imágenes. Se compararon dos algoritmos: LBPH y CNN. Los resultados reflejaron un 95% de precisión al utilizar CNN, lo que llevó a concluir que esta tecnología es adecuada para la gestión de asistencia en clases virtuales.

Mouafo y Biaou (2022) propusieron un modelo de reconocimiento facial apoyado en aprendizaje profundo. El estudio fue aplicado y de diseño experimental, empleando una muestra de 371 imágenes distribuidas en 15 categorías. Los algoritmos utilizados fueron MTCNN para la detección y FaceNet para la extracción de características, evaluando posteriormente modelos de clasificación SVM y KNN. El mejor resultado se obtuvo con KNN, alcanzando un 96,67% de precisión. Se concluyó que la integración de MTCNN y FaceNet, junto con KNN, ofrece un alto desempeño en la identificación de rostros.

Yang y Wei (2021) se enfocaron en analizar la eficacia de la red neuronal convolucional multitarea (MTCNN) en distintos conjuntos de imágenes. La investigación fue de tipo aplicada, con un enfoque cuantitativo. La población se conformó por los datasets CUFS, CUFSF y CASIA NIR-VIS 2.0, sumando más de 11 mil imágenes. Como instrumentos se utilizaron MTCNN y otros algoritmos de comparación. Los resultados evidenciaron niveles de precisión de 98,32%, 97,12% y

96,52% en cada dataset, concluyendo que MTCNN supera a métodos de detección más recientes.

Kolekar y Patil (2023) tuvieron como finalidad implementar un sistema de control de asistencia mediante reconocimiento facial utilizando aprendizaje profundo por transferencia. El estudio fue aplicado y experimental, y se desarrolló con imágenes tomadas en un contexto universitario. Como instrumentos se aplicaron técnicas de deep transfer learning programadas en Python. Los resultados mostraron que el sistema logró registrar la asistencia con alta precisión en tiempo real, concluyendo que estas herramientas fortalecen la automatización en el ámbito educativo.

Huang et al. (2020) buscaron optimizar el reconocimiento facial en escenarios de baja iluminación a través de una red de reconstrucción de características profundas. El estudio, experimental y cuantitativo, utilizó el dataset Specs on Faces (SoF), el cual contiene 42 592 imágenes de 112 sujetos con variaciones de iluminación y oclusiones. Su propuesta permitió mejorar de manera significativa la precisión del reconocimiento en condiciones adversas. Su propuesta permitió mejorar de manera significativa la precisión del reconocimiento en condiciones adversas, evidenciando que la reconstrucción profunda es una estrategia viable para aplicaciones de seguridad y vigilancia.

Zhang et al. (2023) desarrollaron un sistema biométrico multimodal que integró reconocimiento facial y de huellas digitales con el objetivo de reforzar la seguridad. El trabajo, de tipo aplicado, enfoque cuantitativo y diseño experimental, empleó datasets de rostros y huellas como población de análisis, aunque no se especifica un conteo único de imágenes debido a que el estudio trabaja con múltiples fuentes y arquitecturas. Aun así, los experimentos mostraron una precisión superior al

97%, lo que confirma que la combinación de modalidades biométricas incrementa notablemente la fiabilidad respecto a los sistemas unimodales.

Bin-Hamed y Fatnassi (2022) desarrollaron un sistema de registro de asistencia basado en reconocimiento facial empleando una Raspberry Pi como unidad principal de procesamiento. La investigación, de tipo aplicada y con diseño experimental, entrenó el modelo con un conjunto de 77 imágenes organizadas en seis carpetas, mientras que el dispositivo capturaba y procesaba nuevas imágenes en tiempo real. El sistema alcanzó una precisión del 96,7% y redujo en un 48% el tiempo de registro frente al método manual, lo que demuestra la viabilidad de soluciones embebidas de bajo costo para instituciones educativas con recursos limitados.

Azmi et al. (2023) implementaron un sistema inteligente de control de asistencia utilizando reconocimiento facial integrado en una Raspberry Pi 4. El estudio, de carácter aplicado y con enfoque cuantitativo, empleó imágenes capturadas directamente desde una cámara USB conectada al microcomputador. Para el procesamiento facial se aplicaron Haar Cascade y LBPH, logrando una exactitud del 94% en escenarios de iluminación variable. Asimismo, el sistema redujo los errores de registro en un 42%, demostrando que la Raspberry Pi puede ejecutar modelos de visión artificial sin depender de un servidor externo, además de concluir que esta arquitectura embebida facilita aplicaciones de vigilancia y monitoreo educativo con bajo consumo energético.

Chen et al. (2019) propusieron MobileFaceNets, una arquitectura de redes neuronales convolucionales ligeras orientada a la verificación facial en tiempo real en dispositivos móviles. El estudio de tipo experimental, induce el uso de convolución de profundidad global para la mejora de obtención de características faciales con un bajo costo computacional. El modelo fue entrenado con ArcFace sobre el conjunto de datos

MS-Celeb-1M y evaluado en LFW y MegaFace, alcanzando una precisión del 99,55 % en LFW con menos de un millón de parámetros, demostrando así que MobileFaceNets logra alta precisión y rapidez, siendo adecuados para entornos con recursos limitados.

George et al. (2024) propusieron edgeFace, una familia de modelos de reconocimiento facial diseñada para funcionar en dispositivos con recursos limitados. El estudio de enfoque aplicado y diseño experimental, plantea una arquitectura que combina redes neuronales convolucionales con componentes tipo transformer, permitiendo así reducir el costo computacional sin afectar significativamente la precisión. Entre las variantes desarrolladas de edgeFace se encuentran edgeFace-s y edgeFace-xs, las cuales obtuvieron precisiones del 99,78 % y 99,73 % respectivamente en el conjunto de datos LFW. Los resultados lograron evidenciar que edgeFace ofrece un desempeño eficiente y adecuado para aplicaciones de reconocimiento facial en tiempo real de manera viable.

3.2 Bases teóricas

Para entender el desarrollo de la investigación se presentan bases teóricas relacionadas al problema y tecnologías utilizadas para el reconocimiento facial.

3.2.1. *Reconocimiento Facial*

Según Aguileta et al. (2022), el reconocimiento facial es una técnica de identificación que permite autenticar a una persona a partir del análisis de sus rasgos faciales, comparándolos con información previamente almacenada en una base de datos.

Según Zhang et al. (2020), el reconocimiento facial es una tecnología biométrica que permite identificar o verificar a una persona a partir de una imagen

o video. Esta técnica se basa en la detección de características faciales únicas, como la distancia entre los ojos, la forma de la nariz y la estructura ósea del rostro. El reconocimiento facial ha ganado popularidad en aplicaciones de seguridad, control de acceso y registro de asistencia debido a su precisión y eficiencia.

Según Chen et al. (2019), el reconocimiento facial es un procedimiento biométrico el cual permite validar la identidad de una persona mediante la extracción y comparación de características faciales obtenidas a partir de imágenes del rostro. Este proceso se apoya en modelos de aprendizaje profundo, principalmente en redes neuronales convolucionales, las cuales aprenden características faciales relevantes para la identificación.

3.2.2. *Sistemas Embebidos*

De acuerdo con Vahid y Givargis (2001), los sistemas embebidos son plataformas informáticas diseñadas para realizar tareas específicas dentro de un sistema más grande. Estos sistemas suelen integrar hardware y software especializados, como la Raspberry Pi, que es ampliamente utilizada en proyectos de IoT y aplicaciones de bajo costo debido a su flexibilidad y capacidad de procesamiento.

La Raspberry Pi 4B, en particular, destaca por su potencia de procesamiento y su capacidad para ejecutar sistemas operativos completos, lo que la convierte en una opción ideal para implementar aplicaciones de reconocimiento facial en tiempo real, como señala (López , 2017).

3.2.3. H*erramientas*

Según Warden y Situnayake (2019), las librerías de reconocimiento facial, como face_recognition, OpenCV y Dlib, son herramientas esenciales para el desarrollo de aplicaciones de visión por computadora. Estas librerías proporcionan

funciones predefinidas para la detección de rostros, la extracción de características y la comparación de imágenes, lo que simplifica el proceso de desarrollo.

Por otro lado, Python se ha consolidado como el lenguaje de programación preferido para este tipo de proyectos debido a su sintaxis sencilla y a la gran cantidad de librerías disponibles (Lutz, 2009). Además, Python permite una integración eficiente con hardware como la Raspberry Pi, lo que facilita la implementación de sistemas embebidos.

3.2.4. *Sistemas de Control de Asistencia*

Un sistema de control de asistencia registra la entrada y salida del personal para identificar ausencias y retrasos. Puede basarse en tarjetas magnéticas o códigos de barras o en biometría, que ofrece mayor seguridad al reconocer rasgos únicos, aunque con un costo más elevado (Jain et al., 2011).

3.3 Bases conceptuales

Considerando que el elemento principal del proyecto es el “Dispositivo Inteligente basado en reconocimiento facial” que es lo que se estará implementando y evaluando, la variable dependiente refleja el resultado o efecto de dicha implementación. Por lo tanto, se consideran las siguientes variables.

- **Variable independiente**

  "Dispositivo inteligente basado en reconocimiento facial"

- **Variable dependiente**

  "Sistema de registro de asistencia"

3.3.1. *Evaluación de la variable dependiente*

Tabla 1

Operacionalización de las variables

Variable	Dimensión	Indicadores	Medida
Sistema de registro de asistencia	Rendimiento	Exactitud	Porcentaje
		Precisión	Porcentaje
		Sensibilidad	Porcentaje
		Especificidad	Porcentaje

Nota. Elaboración propia.

Métricas de Evaluación del Sistema de Reconocimiento Facial

En el contexto del reconocimiento facial, la matriz de confusión es una herramienta fundamental para evaluar el rendimiento de los sistemas de identificación. Según Vakili et al. (2020), la evaluación del rendimiento de un sistema basado en aprendizaje automático (machine learning) se fundamenta en la matriz de confusión, una herramienta que permite analizar la capacidad de un modelo para clasificar correctamente las instancias. A partir de esta matriz, se derivan indicadores clave como la exactitud, precisión, sensibilidad y especificidad, las cuales son esenciales para comprender el comportamiento del sistema en tareas de clasificación.

- **Verdaderos Positivos (VP):**

Número de asistencias correctamente registrados en el sistema del dispositivo, así como en la hoja de control. Es decir, el usuario asistió, marcó en la hoja de control y el sistema lo reconoció correctamente.

- **Verdaderos Negativos (VN):**

  Número de registros, en donde el usuario no asistió, por ende, no registró en la hoja de control, y el sistema lo marcó como ausente.

- **Falsos Positivos (FP):**

  Número de registros erróneos, en donde el usuario aparece registrado en la hoja, pero no aparece en el sistema. El sistema emitió un resultado negativo.

- **Falsos Negativos (FN):**

  Número de registros erróneos, en donde el usuario aparece registrado en el sistema, pero no aparece en la hoja. El sistema emitió un resultado positivo.

Tabla 2

Matriz de confusión

	Clase	**Predicción del modelo**	
		Positivo	**Negativo**
Resultado Real	Positivo	Verdadero Positivo (VP)	Falso Negativo (FN)
	Negativo	Falso Positivo (FP)	Verdadero Negativo (VN)

Nota. Elaboración propia.

A continuación, se definen los indicadores de rendimiento del sistema:

- **Exactitud (Acurracy)**

Mide la proporción de registros, tanto de asistencia como de ausencia, que el sistema clasificó correctamente respecto al total de registros analizados. En el contexto del sistema de asistencia, un registro es correcto cuando el dispositivo coincide con la hoja física, independientemente de si el usuario estaba presente o ausente.

$$Acurracy = \frac{VP + VN}{VP + FP + FN + VN} \quad (1)$$

- **Sensibilidad (Recall)**

  Mide la capacidad del sistema para registrar correctamente a los usuarios que sí asistieron. Responde a la pregunta: de todos los días en que un usuario estuvo presente según la hoja física, ¿en cuántos el dispositivo lo registró exitosamente? Un valor bajo indica que el sistema "pierde" asistencias reales.

$$Recall = \frac{VP}{VP + FN} \quad (2)$$

- **Precisión**

  Mide qué tan confiables son los registros de presencia generados por el sistema. Responde a la pregunta: de todos los días en que el dispositivo marcó a un usuario como presente, ¿cuántos correspondieron realmente a una asistencia real según la hoja física? Un valor bajo indica que el sistema genera registros de presencia falsos.

$$Precisión = \frac{VP}{VP + FP} \quad (3)$$

- **Especificidad**

  Mide la capacidad del sistema para no registrar a usuarios que estuvieron ausentes. Responde a la pregunta: de todos los días en que un usuario no asistió según la hoja física, ¿en cuántos el dispositivo correctamente omitió registrarlo? Un valor bajo indica que el sistema marca presentes a personas que no asistieron.

$$Especificidad = \frac{VP}{VN + FP} \quad (4)$$

4. DISEÑO METODOLÓGICO

De acuerdo con Arias (2021), el estudio aplicado se concentra en solucionar problemas específicos de la realidad mediante la utilización de conocimientos existentes, con el objetivo de ofrecer soluciones prácticas y útiles en contextos concretos. Bajo este contexto, el estudio actual se categoriza como una investigación aplicada, debido a que se orienta a la solución de un problema concreto mediante el desarrollo de un dispositivo inteligente para el registro de asistencia; no se limita a la generación de conocimiento teórico, sino que busca su implementación práctica. Asimismo, integra tecnologías existentes como el reconocimiento facial en un contexto real; por tanto, su finalidad es producir un impacto directo en el ámbito operativo.

Por otra parte, Hinojosa et al. (2024) definen el enfoque cuantitativo como aquel que utiliza la compilación y examen de registros cuantitativos para detectar patrones, validar teorías y responder preguntas de investigación de forma objetiva. En este sentido, el presente estudio adopta un enfoque cuantitativo ya que se basa en la recolección y análisis de datos medibles relacionados con el rendimiento del sistema, se emplean métricas como precisión tiempo de respuesta y tasa de reconocimiento para evaluar el dispositivo, los resultados se procesan mediante métodos estadísticos que permiten validar objetivamente su eficacia, garantizando así la objetividad y replicabilidad de los hallazgos.

Según Arispe et al. (2020), el diseño experimental implica la manipulación controlada de una o más variables, desarrollándose en un entorno regulado que permita garantizar la validez de los resultados obtenidos. Bajo este contexto, el presente estudio emplea un diseño experimental, puesto que implica la manipulación de variables para evaluar el comportamiento del dispositivo en condiciones controladas, se realizan pruebas específicas para medir el desempeño del sistema de reconocimiento facial,

además se comparan resultados bajo distintos escenarios de uso, lo que permite establecer relaciones causales entre las variables analizadas.

4.1 Diseño de contrastación y procedimiento

4.1.1. *Diseño de contrastación de hipótesis*

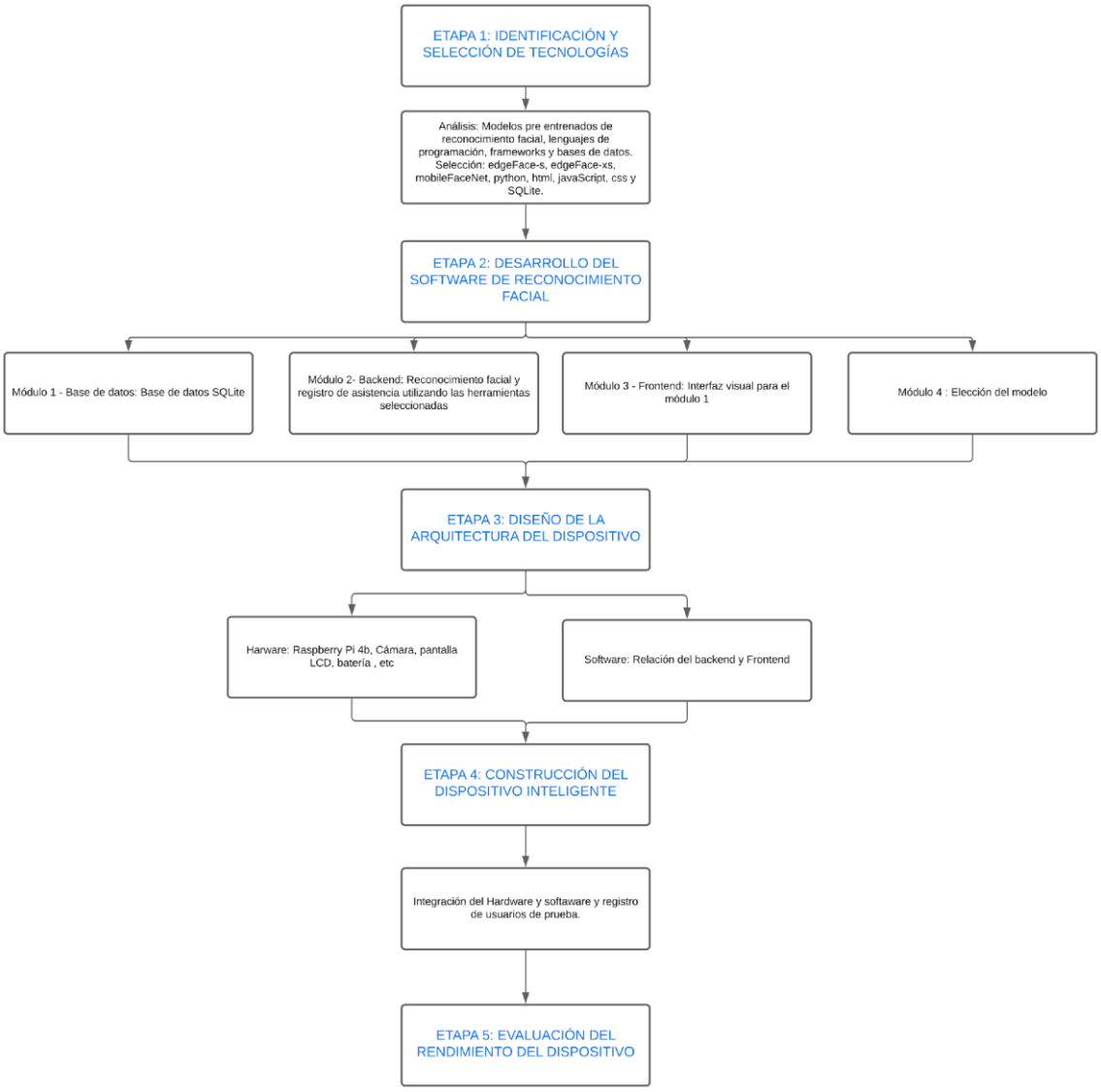
Al ser un trabajo tecnológico, la siguiente investigación no plantea una hipótesis.

4.1.2. *Procedimiento a seguir en la investigación*

El procedimiento para la implementación del Dispositivo Inteligente basado en reconocimiento facial para el registro de asistencia se estructuró en 5 etapas, cada una con la intención de cumplir los objetivos específicos planteados. Esta estructura de muestra a continuación, en la Figura 1.

Figura 1

Procedimiento



Nota. Elaboración Propia.

Como se muestra en la figura 1, **la primera etapa** consistió en realizar un análisis de distintas tecnologías tratando de que sean las más eficientes en entornos con bajos recursos de hardware, teniendo en cuenta que trabajamos con una Raspberry Pi 4b, es así que se seleccionaron modelos pre entrenados que se utilizan para el reconocimiento facial y detección de rostros.

Se evaluaron varios modelos, pero de entre todos se seleccionaron 3, estos fueron edgeFace-s, edgeFace-xs y mobileFaceNet, pues estos modelos están optimizados para dispositivos “edge”, es decir, con recursos de hardware limitados. Estos modelos generan vectores de características o embeddings tomando como base una imagen de un rostro, los modelos de tipo edgeFace producen vectores de 512 dimensiones, mientras que el mobileFaceNet genera embeddings de 128 dimensiones. Estos modelos se evaluarán de mejor manera en la etapa de desarrollo de software.

Con respecto a las tecnologías de desarrollo de software se optó por elegir Python como lenguaje para el backend y FastApi como el framework del mismo. Se eligió Python porque este es el lenguaje que mejor se adapta con dispositivos que tienen sistemas operativos en base a Linux, y FastApi por ser un framework que no consume muchos recursos siendo el mejor para este caso. Para el desarrollo del frontend solo se eligieron tecnologías bases como HTML, JavaScript y CSS, sin tener en cuenta un framework, puesto que restaría potencia al Raspberry Pi.
Considerando que se necesitará almacenar datos de reconocimiento y del registro de asistencia, se evaluó utilizar SQLite como un gestor de base de datos óptimo para el almacenamiento de datos.

La evaluación se basó tomando en cuenta varios criterios, como los recursos de hardware que se tienen, criterios de precisión en el reconocimiento de rostros, velocidad de procesamiento, la complejidad de implementación de estas tecnologías. Como resultado de este análisis se optó por trabajar en conjunto con las tecnologías mencionadas, pues se adaptan a las necesidades y limitaciones del proyecto.

La segunda etapa se centró en desarrollar el software de reconocimiento facial, que está separado en 3 módulos*:*

- *El primer módulo* corresponde a al diseño de la base de datos, la que se usó es SQLite, que es una base de datos embebida y ligera, esta fue óptima en la placa Raspberry PI. Se almacenaron los patrones faciales de los usuarios registrados, los datos personales de los mismos, y registros de asistencia por día estado una vez el dispositivo esté en funcionamiento.
- *El segundo módulo* es el Backend del sistema, que gestionó el registro de usuarios, la detección de rostros, y el registro de asistencia. Con respecto al registro de usuarios se procesó imágenes en base 64 de los usuarios que se estén por registrar, un aproximado de 8 imágenes convertidas por usuario, estos datos fueron procesados por las diferentes librerías de extracción de características faciales, y estos patrones fueron almacenados en la base de datos, para su posterior uso en el reconocimiento de usuarios registrados. Con respecto al registro de asistencia se tomaron en cuenta que el usuario reconocido pasó primero por la fase de reconocimiento, luego se registró su asistencia en la base de datos tomando en cuenta la hora exacta y validando el estado de la misma, que pueden ser “Presente”, “Tardanza” o “Falta” si es que el reconocimiento se hizo fuera del horario permitido.
- *El tercer módulo* fue el frontend del sistema, que se encargó de mandar los datos hacia el backend debidamente procesados, como las imágenes en base64, los datos personales previamente validados, y los datos del reconocimiento del usuario para registrar la asistencia.

La tercera etapa fue el diseño de la arquitectura del dispositivo, acá se tomó en cuenta la relación entre componentes de hardware y software, su

funcionamiento en conjunto, posibles mejoras para un mejor desempeño, y la batería que se utilizó para levantar todo el dispositivo sin problemas.

La cuarta etapa fue la construcción física del dispositivo, tomando en cuenta el diseño previamente construido, de tal modo de que funcionó como se planeó, se verificó el funcionamiento de cada componente de hardware como el Raspberry usado, la cámara conectada, la pantalla que sirvió para mostrar la interfaz y demás detalles. Se corroboró que los 3 componentes de software funcionen en sincronía y así se pueda ya comenzar a registrar los usuarios para las posteriores pruebas. Con respecto al registro de usuarios de prueba se consideró en esta etapa porque fue necesario la construcción completa del dispositivo para poder evaluarlo, se registraron un aproximado de 10 imágenes por cada usuario desde la interfaz para su luego procesamiento.

Finalmente, **la quinta etapa** fue la evaluación de del rendimiento del dispositivo a través de diferentes pruebas en condiciones de uso real, tomando en cuenta todos los escenarios posibles, y diferentes condiciones de luz, distancia de la cámara, así como diferentes expresiones de los usuarios al registrar su asistencia. Se establecieron métricas cuantitativas para determinar la efectividad del dispositivo. Estos resultados fueron documentados y analizados para determinar si es efectivo el dispositivo, y su posible implementación en entornos reales.

4.2 Población, muestra

La población para este estudio fue el equipo GICDIAC – Grupo de Investigación en Ciencia de Datos, Inteligencia Artificial y Ciberseguridad; mientras que la muestra está conformada por un grupo de 10 personas

pertenecientes al área de sistemas, quienes fueron registradas en el dispositivo para las pruebas de evaluación.

Para cada uno de los 10 usuarios se capturaron 8 imágenes faciales mediante la interfaz de registro del dispositivo, considerando distintas expresiones faciales (rostro neutro, sonrisa, boca cerrada) y leves giros de cabeza hacia la derecha e izquierda, con el objetivo de generar un perfil facial robusto que represente adecuadamente al usuario en condiciones reales de uso.

4.3 Técnicas, instrumentos, equipos y materiales

Las técnicas de recolección de datos empleadas en la investigación son:

- Análisis documentario, que permitió identificar los algoritmos más eficientes para un sistema de control de asistencia.
- Captura de imágenes, utilizada para recopilar fotografías de los usuarios (dataset) destinadas al entrenamiento y validación del modelo.
- Amplificación de imágenes, aplicada para incrementar el dataset con fines de mejorar el entrenamiento del modelo.

Los instrumentos utilizados para cada técnica fueron:

- Para el análisis documentario: bases de datos científicas como *ScienceDirect, IEEE Xplore, SCOPUS y EBSCO*, priorizando artículos de los últimos cinco años.
- Para la captura de imágenes: la cámara web de 48 MP integrada en una laptop.

En cuanto a los equipos y materiales empleados en el procesamiento y análisis de datos, se utilizaron:

- Laptop MSI, con procesador Ryzen 7 y 16 GB de RAM.

- Lenguaje de programación Python v.3.10, en el entorno Anaconda Navigator mediante Jupyter Notebook.

5. RESULTADOS

5.1 Resultado 1: Identificación y selección de herramientas para el proyecto

Se realizó un análisis comparativo de tecnologías que se podían adaptar a las necesidades del proyecto. Esta comparación consideró cuatro aspectos fundamentales: compatibilidad con componentes de hardware limitado en potencia, precisión de detección y reconocimiento, velocidad de procesamiento, y facilidad de implementación.

Elección de microcontrolador

Considerando lo mencionado en el párrafo anterior, se eligió al Raspberry Pi 4B como el núcleo del proyecto, pues se adaptaba tanto a las necesidades como a los recursos disponibles, además de que brinda la potencia necesaria que requiere el proyecto, en comparación con otras alternativas; ver tabla 3.

Tabla 3.

Comparación de microcontroladores para el dispositivo

Micro controladores	Compatibilidad con hardware limitado	Precisión (soporte IA)	Velocidad de procesamiento	Facilidad de implementación	Evaluación general
Raspberry Pi 4B	Alta	Alta	Media-Alta	Alta	**Óptima**
Arduino Uno	Muy alta	Baja	Baja	Alta	Limitada
ESP32-CAM	Alta	Baja	Baja	Media	Limitada

Nota. Elaboración propia

En la Tabla 3, se presenta la comparación entre distintos microcontroladores considerando criterios como compatibilidad con recursos limitados, capacidad de procesamiento, soporte para aplicaciones de inteligencia artificial y facilidad de

implementación, en donde se observa que la Raspberry Pi 4B obtiene una valoración global óptima debido a su equilibrio entre rendimiento y bajo consumo, a diferencia de alternativas como Arduino Uno o ESP32-CAM que presentan limitaciones en procesamiento.

Elección del gestor de base de datos

Con respecto al gestor de base de datos, se eligió SQLite por su capacidad de poder estar embebida, y además de su gran ligereza, que son las características deseadas al trabajar con recursos limitados como el Raspberry Pi 4B. Esta base de datos no requiere contar con un servidor separado o tener una instancia fuera del proyecto, y puede funcionar directamente desde el sistema de archivos del proyecto, logrando así mejorar el consumo de memoria y procesamiento. El análisis y comparación con otros gestores de base de datos se muestra en la Tabla 4.

Tabla 4

Comparación de gestores de bases de datos

Característica	**SQLite**	**MySQL**	**PostgreSQL**
Tipo de arquitectura	Embebida	Cliente-servidor	Cliente-servidor
Consumo de recursos	Muy bajo	Medio	Medio-Alto
Instalación	No requiere	Requiere	Requiere
Administración	Simple	Moderada	Compleja
Escalabilidad	Baja	Alta	Muy alta
Adecuado para sistemas IoT	Muy adecuado	Poco adecuado	Poco adecuado

Nota. Elaboración propia

En la Tabla 4 se realiza una comparación específica entre SQLite y otros sistemas de gestión de bases de datos, destacándose que SQLite posee una arquitectura embebida que minimiza el consumo de recursos y simplifica su implementación, lo cual resulta altamente adecuado para dispositivos IoT como el desarrollado en la investigación,

mientras que alternativas como MySQL y PostgreSQL, aunque más robustas y escalables, no resultan eficientes para entornos con limitaciones de hardware.

Elección de modelos de reconocimiento facial

En cuanto a los modelos para reconocimiento facial, según la literatura actual investigada, existen tres mejores modelos que pueden ser utilizados en esta investigación, con la finalidad de obtener resultados más precisos y óptimos. La descripción de cada uno de ellos se realiza a continuación:

- **edgeFace-s**

  Fue seleccionado como uno de los modelos principales para el proceso de reconocimiento facial debido a su alto equilibrio entre precisión y eficiencia computacional. Esta variante está diseñada para operar en dispositivos con recursos limitados, manteniendo un desempeño competitivo en tareas de verificación facial. EdgeFace-S emplea una arquitectura híbrida que combina redes neuronales convolucionales y componentes tipo Transformer, lo que permite una adecuada extracción de características faciales con un menor consumo de memoria y tiempo de procesamiento.

- **edgeFace-xs**

  Fue incorporado por su ligereza y bajo costo computacional, siendo adecuado para escenarios donde existen mayores restricciones de hardware. A pesar de su menor tamaño, este modelo conserva una precisión elevada en conjuntos de datos de referencia, lo que lo convierte en una alternativa eficiente para aplicaciones de reconocimiento facial en tiempo real. Su diseño prioriza la optimización de recursos sin comprometer de forma significativa la calidad del reconocimiento.

- **mobilefaceNet**

  Fue seleccionado por ser un modelo adecuado para entornos con recursos limitados, como los dispositivos móviles y los sistemas embebidos. Se caracteriza por emplear una arquitectura compacta basada en redes neuronales convolucionales, la cual optimiza el uso de los parámetros del modelo sin sacrificar el desempeño. Además, incorpora estrategias que permiten obtener descriptores faciales representativos de manera eficiente, lo que facilita su aplicación en tareas de reconocimiento facial en tiempo real, manteniendo un rendimiento estable y tiempos de respuesta reducidos.

La elección final del modelo a utilizar se realizará en el desarrollo de los siguientes objetivos específicos.

Lenguaje de programación

Se eligió Python como lenguaje de programación base para el desarrollo, porque es el lenguaje por excelencia para este tipo de proyectos, que combinan visión por computadora y aprendizaje automático, además de integrarse perfectamente con el SO del Raspberry Pi utilizado.

Esta combinación de tecnologías proporciona un ambiente robusto que ayudará a construir un proyecto eficiente y escalable.

5.2 Resultado 2: Desarrollo del software incorporando herramientas de reconocimiento facial.

Para poder desarrollar el software, se estructuró su implementación en diversas etapas que garantizan su correcta funcionalidad y coherencia técnica, en primer lugar se establecen los requerimientos del sistema con el fin de definir las funcionalidades y restricciones que orientan su desarrollo, posteriormente se presenta el diseño del diagrama entidad relación de la base de datos que sustenta el almacenamiento de la información, seguido de la descripción de la estructura del proyecto de software basada en un enfoque modular, a continuación se detalla el desarrollo de las interfaces gráficas que permiten la interacción con el usuario, y finalmente se expone el proceso de selección del modelo de reconocimiento facial más adecuado en función de su rendimiento y eficiencia en el entorno propuesto.

5.2.1 Requerimientos del software

A continuación se presentan los requerimientos del sistema desarrollado, dado que la definición de estos constituye una etapa fundamental dentro del desarrollo del software, ya que permite establecer de manera clara las funcionalidades y restricciones que guiarán las fases posteriores del proyecto, en ese sentido, se identificaron los requerimientos funcionales, los cuales describen las operaciones y servicios que el sistema debe cumplir en función de las necesidades del registro de asistencia mediante reconocimiento facial, así como los requerimientos no funcionales, orientados a garantizar aspectos de calidad como rendimiento, eficiencia y usabilidad, los cuales en conjunto aseguran el correcto funcionamiento del sistema desarrollado; ver Tabla 5 y 6.

Tabla 5.

Requerimientos funcionales

Código	Requerimiento funcional	Descripción
RF-01	Gestionar grupos por modelo	El sistema debe permitir crear y administrar grupos asociados a un modelo específico, garantizando la organización jerárquica de los usuarios
RF-02	Registrar usuarios	El sistema debe permitir registrar nuevos usuarios mediante un formulario de datos personales y su asociación a un grupo determinado
RF-03	Capturar imágenes del usuario	El sistema debe permitir capturar entre 5 y 10 imágenes faciales por usuario desde diferentes ángulos para la construcción del dataset de entrenamiento
RF-04	Procesar y normalizar encodings faciales	El sistema debe detectar el rostro, extraer las 128 mediciones faciales y normalizar los vectores de características para su almacenamiento
RF-05	Almacenar encodings en la base de datos	El sistema debe guardar los embeddings faciales en la base de datos vinculándolos con el modelo, grupo y usuario correspondiente
RF-06	Realizar reconocimiento facial en tiempo real	El sistema debe capturar video en tiempo real, detectar rostros en cada frame y compararlos con los registros almacenados para identificar usuarios
RF-07	Validar la identidad del usuario	El sistema debe determinar si la similitud facial supera el umbral establecido para emitir una respuesta positiva o negativa de reconocimiento
RF-08	Registrar asistencia automáticamente	El sistema debe registrar la asistencia del usuario identificado en función de la hora de reconocimiento y asignar el estado correspondiente
RF-09	Evitar registros duplicados	El sistema debe impedir que un mismo usuario genere más de un registro de asistencia en el mismo día
RF-10	Visualizar información operativa en tiempo real	El sistema debe mostrar la vista de cámara, el estado del sistema y las estadísticas diarias de asistencia en la interfaz principal
RF-11	Gestionar reportes de asistencia	El sistema debe permitir consultar el listado de asistencias por fecha y exportar la información en diferentes formatos
RF-12	Gestionar cámaras de captura	El sistema debe permitir seleccionar y enlazar una cámara para el proceso de registro y reconocimiento facial

RF-13	Ejecutar pruebas comparativas de modelos	El sistema debe permitir evaluar distintos modelos faciales para verificar su desempeño en escenarios de prueba
RF-14	Validar los flujos del sistema	El sistema debe mostrar mensajes de validación ante errores o procesos exitosos

Nota. Elaboración propia.

Tabla 6

Requerimientos no funcionales

Código	Requerimiento no funcional	Descripción
RNF-01	Rendimiento	El sistema debe ejecutar el proceso de registro facial en un tiempo aproximado de 30 a 40 segundos por usuario, sin afectar de forma crítica la operación general
RNF-02	Procesamiento en tiempo real	El sistema debe mantener la captura y análisis de video con una tasa cercana a 30 FPS, en función de las condiciones de hardware disponibles
RNF-03	Eficiencia de recursos	El software debe operar con bajo consumo de memoria y procesamiento, considerando que será ejecutado en hardware de capacidad limitada como Raspberry Pi 4B
RNF-04	Integridad de datos	La base de datos debe conservar relaciones consistentes entre modelo, grupo, usuario, encodings y asistencia, evitando registros incongruentes.
RNF-05	Usabilidad	La interfaz debe presentar menús y formularios intuitivos que faciliten el registro, la consulta y la validación de asistencia por parte del usuario final
RNF-06	Fiabilidad	El sistema debe garantizar un comportamiento estable en el reconocimiento facial y en el registro de asistencia bajo condiciones operativas normales
RNF-07	Compatibilidad	El software debe ser compatible con el entorno de ejecución definido para el proyecto, incluyendo la arquitectura MVC y la base de datos SQLite
RNF-08	Mantenibilidad	La estructura del sistema debe permitir su actualización, corrección y ampliación mediante una arquitectura modular y separada por componentes

RNF-09	Seguridad lógica	El sistema debe restringir el acceso a funciones críticas, proteger la información de usuarios y evitar alteraciones no autorizadas sobre los registros
RNF-10	Portabilidad	El software debe poder ejecutarse en un entorno embebido sin requerir una infraestructura de servidor separada
RNF-11	Disponibilidad operativa	El sistema debe permanecer accesible durante la jornada de registro de asistencia, permitiendo el uso continuo de cámara, y base de datos.

Nota. Elaboración propia.

5.2.2. Diagrama entidad relación de la BD.

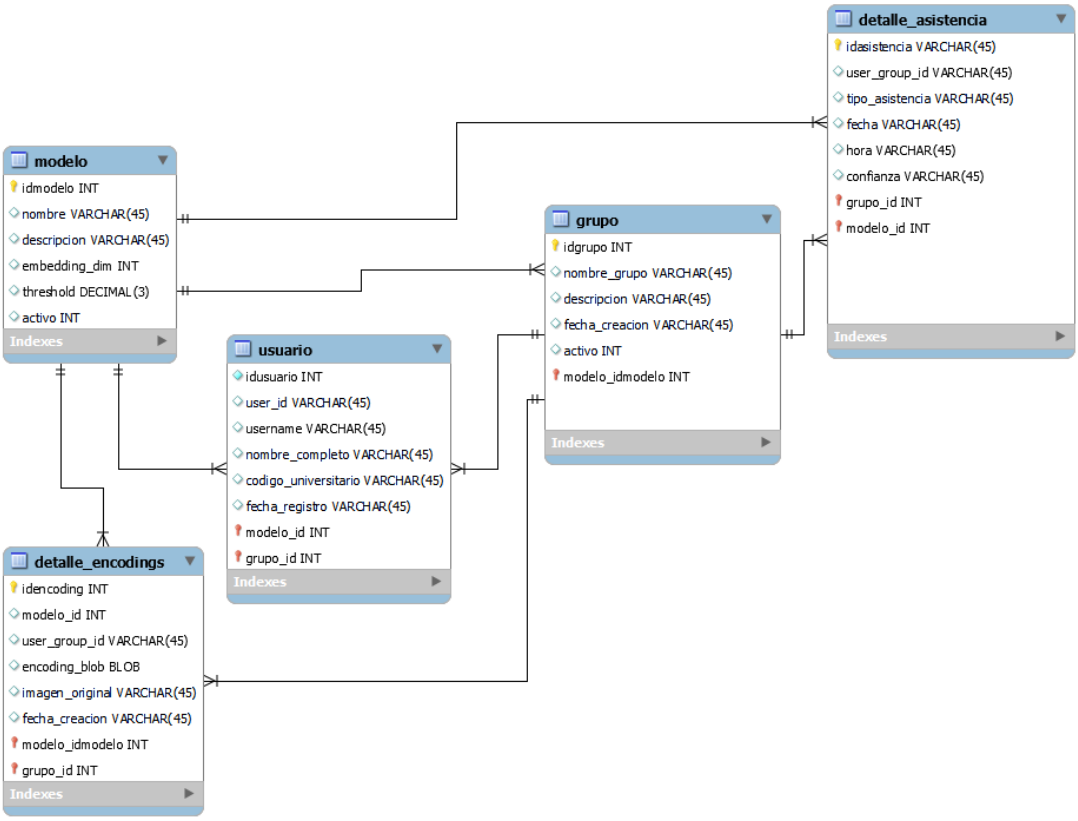
Luego de realizar el análisis de requerimientos de información, se consideraron las entidades pertinentes y necesarias para que el sistema se pueda desarrollar de la mejor manera; dichas entidades posteriormente se convertirán en tablas dentro de la base de datos, las cuales se mencionan a continuación:

- "modelo"
    - Propósito: Almacenar los modelos de reconocimiento facial.
- "grupo"
    - Propósito: Almacenar grupos de un modelo en específico.
- "usuario"
    - Propósito: Almacenar información de los usuarios pertenecientes a un grupo y modelo específico.
- "detalle_encodings"
    - Propósito: Almacenar los embeddings de los rostros capturados.
- "detalle_asistencia"
    - Propósito: Registrar cada evento de reconocimiento / asistencia que se llegue a detectar.

Con las entidades ya identificadas, se procedió a crear el diagrama entidad relación considerando los atributos a cada entidad respectiva. Además, considerando que la base de datos fue de tipo relacional, el diagrama entidad-relación quedó tal cual como se muestra en la Figura 2.

Figura 2

Diagrama ER del proyecto



Nota. Elaboración propia.

Las relaciones entre las tablas de la base de datos, es fundamental en el momento de realizar los registros en cada una de ellas, por tal motivo, dichas relaciones se comentan a continuación:

- Modelo (1) --- (N) Grupo

  - Cada grupo debe estar asociado a exactamente 1 modelo

  - Un modelo puede tener muchos grupos

- Grupo (1) --- (N) Usuario

- Un usuario pertenece a exactamente 1 grupo

- Un grupo contiene muchos usuarios

- Usuario (1) → detalle_encodings

  - Cada encoding corresponde a exactamente 1 modelo, 1 grupo y 1 usuario

  - Un grupo puede tener muchos encodings (múltiples fotos por usuario)

- Modelo - grupo - usuario → detalle_asistencia (3)

  - Cada registro de asistencia usa exactamente 1 modelo, ocurre en 1 grupo y se relaciona con 1 usuario.

  - Un grupo acumula muchos registros de asistencia

Con todo lo desarrollado, se logró cubrir toda la necesidad de almacenamiento de información, además de poder escalar sin ningún problema, pues, soporta múltiples modelos de reconocimiento facial, garantizando poder crear los grupos que se requieran por cada modelo, y por defecto, guardar múltiples usuarios manteniendo una correcta separación.

5.2.3. Programación del software

Para realizar la programación mediante código, primero se definió la estructura del proyecto, organizando el código de forma clara para facilitar su comprensión y mantenimiento. A continuación, se presenta un gestor de archivos con la estructura general del sistema:

```
proyecto/
├── api/                    # API REST - FastAPI
│   ├── routes/             # Rutas organizadas por dominio
│   │   ├── attendance.py   # Gestión de asistencias
│   │   ├── datasets.py     # Captura de datasets
│   │   ├── grupos.py       # CRUD de grupos
```

```
│   │   ├── model_switch.py       # Cambio dinámico de modelos
│   │   └── users.py              # Gestión de usuarios
│   └── _init_.py
├── core/                         # Lógica de negocio
│   ├── attendance_service.py     # Servicio de asistencias
│   ├── enrollment_service.py     # Registro de usuarios
│   ├── model_manager.py          # Gestor de modelos (Singleton)
│   └── websocket_handler.py      # WebSocket para reconocimiento en tiempo real
├── database/                     # Persistencia de datos
│   ├── db_connector.py           # Conector SQLite
│   ├── db_helper.py              # Inicialización de BD
│   └── db_models.py              # Esquemas y modelos de datos
├── models/                       # Modelos de reconocimiento
│   ├── base_recognizer.py        # Clase abstracta
│   ├── edgeface/                 # EdgeFace (s y xs)
│   │   ├── recognizer.py
│   │   ├── recognizer_xs.py
│   │   └── README.md
│   ├── mobilefacenet/            # MobileFaceNet
│   │   └── recognizer.py
│   └── weights/                  # Pesos de los modelos (.onnx)
├── public/                       # Frontend (SPA)
│   ├── css/                      # Estilos organizados
│   │   ├── main.css
│   │   ├── navbar.css
│   │   ├── content.css
│   │   ├── modal.css
│   │   └── welcome.css
│   ├── js/                       # Lógica del cliente
│   │   ├── helpers/
```

```
│   │   │   ├── helper.js
│   │   │   └── modal.js
│   │   ├── app.js              # Configuración global
│   │   ├── navbar.js           # Navegación
│   │   ├── home.js             # Reconocimiento (modelos)
│   │   ├── home-edgeface.js    # Reconocimiento (EdgeFace-xs)
│   │   ├── usuarios.js         # Gestión de usuarios
│   │   ├── settings.js         # Registro de usuarios
│   │   ├── datasets.js         # Captura masiva de imágenes
│   │   ├── asistencia.js       # Asistencias
│   │   └── grupos.js           # Gestión de grupos
│   └── index.html              # Punto de entrada del frontend
├── main.py                     #Punto de entrada del sistema
└── requirements.txt            # Dependencias
```

A partir de esta estructura, el desarrollo del software se organizó en tres módulos principales, definidos de acuerdo con el orden seguido durante su implementación y las funcionalidades del sistema.

Módulo 1*:* Backend de Procesamiento

En el backend se desarrolló la implementación de las siguientes funcionalidades principales:

Registro de Usuarios:

El proceso de registro captura 8 imágenes de cada usuario desde diferentes ángulos (frontal, laterales, con ligeras inclinaciones o expresiones de rostro). Cada imagen se procesa así:

- Detección del rostro
- Extracción de 512 mediciones faciales mediante el modelo seleccionado
- Normalización de los encodings

- Almacenamiento en la base de datos

Este proceso tarda aproximadamente 30-40 segundos por usuario y genera un perfil facial robusto que se usará en el futuro reconociendo de usuarios registrados.

Reconocimiento Facial en Tiempo Real:

- Captura los frames de video a 30 FPS en promedio, porque puede variar.
- Detecta rostros en cada frame mediante los algoritmos de las librerías utilizadas.
- Extrae los encodings faciales de los rostros detectados
- Compara cada encoding con los almacenados en la base de datos usando distancia euclidiana.
- Identifica al usuario si la similitud supera el umbral establecido (0.6 por defecto).

Luego de este proceso, se envía una respuesta, que puede ser positiva o negativa, dependiendo si reconoció al usuario.

Gestión de Asistencia:

Si el usuario es identificado:

- Se verifica la hora del registro
- Asigna el estado correspondiente a la hora (presente, tardanza o falta)
- Se toma en cuenta que no haya registros duplicados en ese día.

Módulo 2: Interfaz visual

La interfaz visual está estructurada por menús, estos cumplen responsabilidades separadas; por un lado, se tiene los menús para el resultado final, y por el otro, se tiene los menús para el registro de datasets para poder evaluar los modelos seleccionados. Es así que se planteó definir la interfaz principal en este módulo, mientras que los otros menús se explican en el módulo 3.

- **Interfaz Principal:**

Figura 3

Interfaz de inicio del sistema

- La opción “Iniciar sistema” permite el ingreso al sistema desarrollado

Figura 4

Interfaz principal del sistema

- Vista de cámara en tiempo real con cuadro de detección de rostros
- Indicadores visuales del estado del sistema
- Estadísticas de las asistencias del día

- **Panel de registro:**

Figura 5

Interfaz del Registro de Usuario

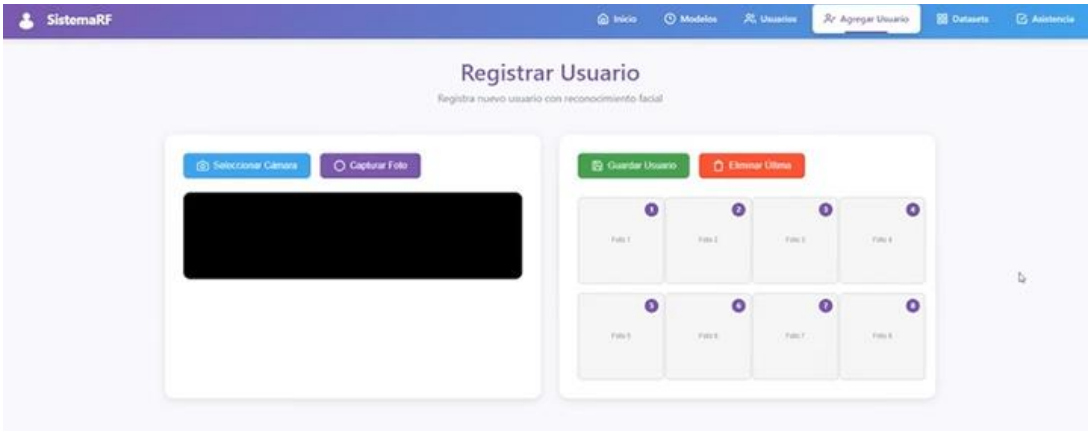
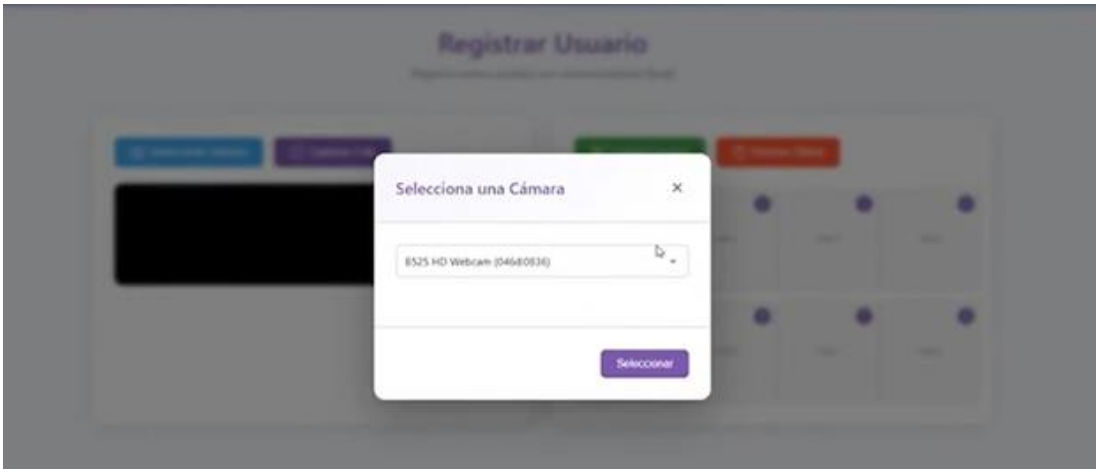


Figura 6

Formulario para Registro



- Formulario de registro para nuevos usuarios
- Captura de fotos del usuario en cuestión

- **Panel de Reportes:**
  - Listado de asistencias por fecha
  - Exportación de datos en diferentes formatos

Se utilizó una arquitectura MVC (Modelo-Vista-Controlador) para integrar los módulos desarrollados.

Módulo 3: Elección del modelo

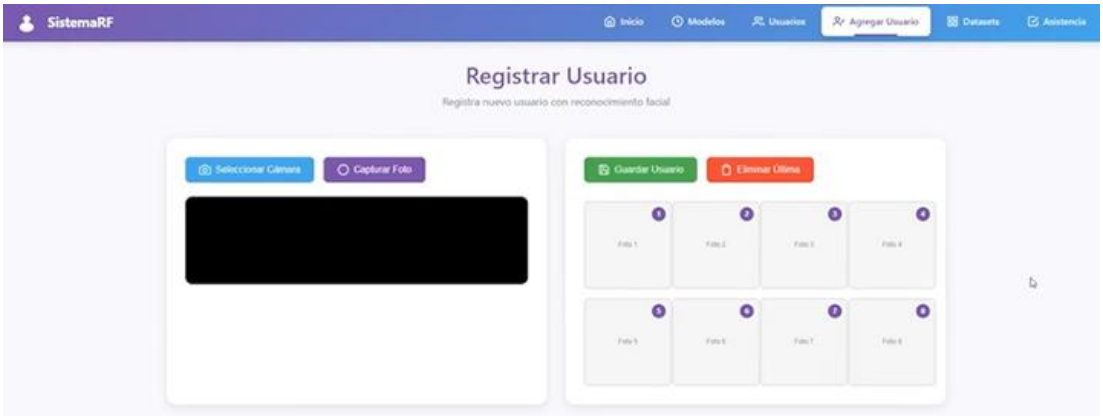
Se diseñó una interfaz conjunta al principal, teniendo en cuenta su uso durante la selección del modelo más conveniente. Algunos de los menús implementados forman parte de la interfaz definitiva del sistema, mientras que otros se incluyeron únicamente para realizar las pruebas y validar el funcionamiento, por lo que no se consideran dentro de la versión final del sistema.

A continuación, se describió el proceso seguido a través explicaciones concisas.

Registro de datasets

Figura 7

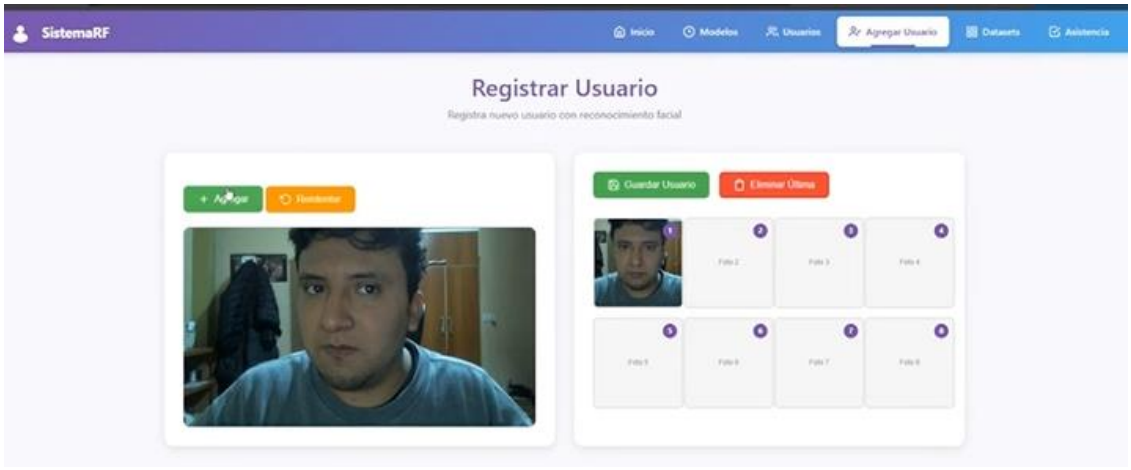
Registro de Usuario



- Para el registro de usuario, se seleccionó “Capturar Foto”, para capturar 8 imágenes del nuevo usuario en diferentes ángulos.

Figura 8

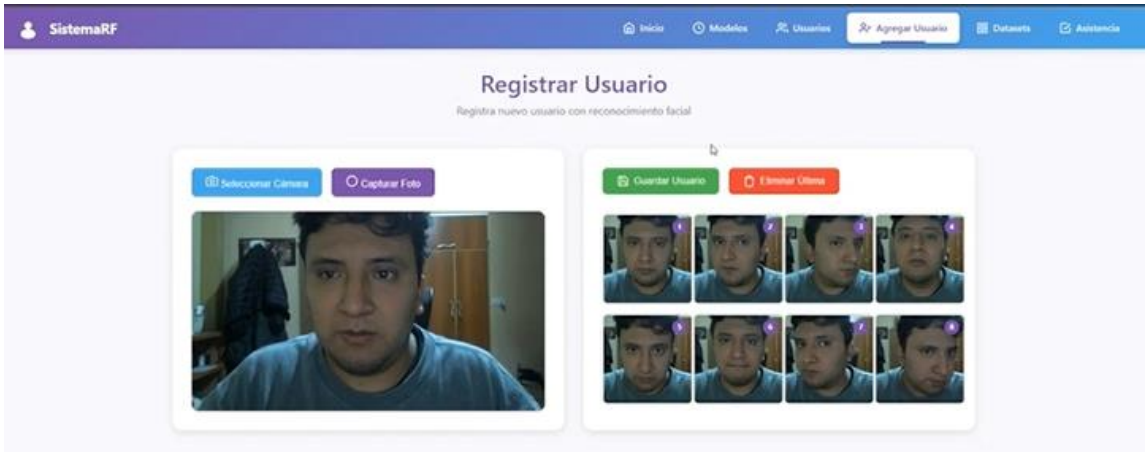
Inicio - Captura de Imágenes



- Se seleccionó el botón (+Agregar), añadiendo inmediatamente imagen por imagen a la tabla de imágenes vacías

Figura 9

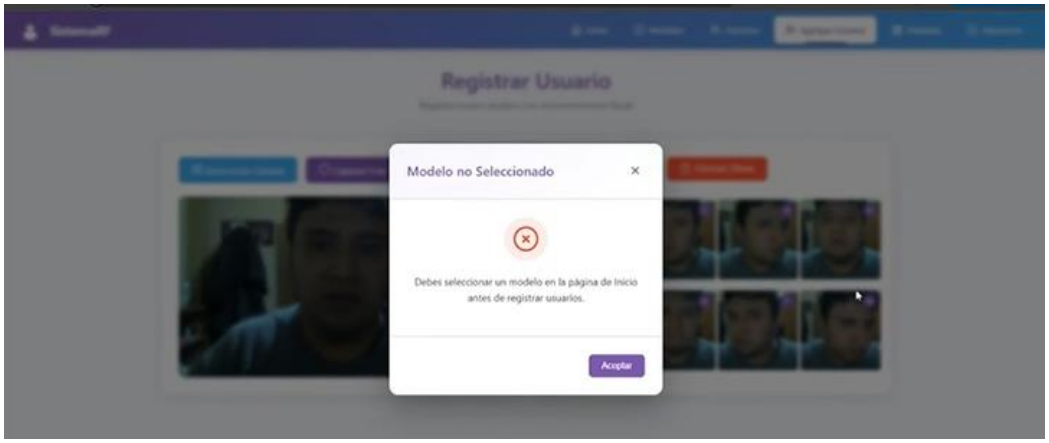
Fin - Captura de imágenes



- Se completó la tabla de imágenes exitosamente.
- Seleccionó el botón “Guardar Usuario”, para un registro válido del usuario.

Figura 10

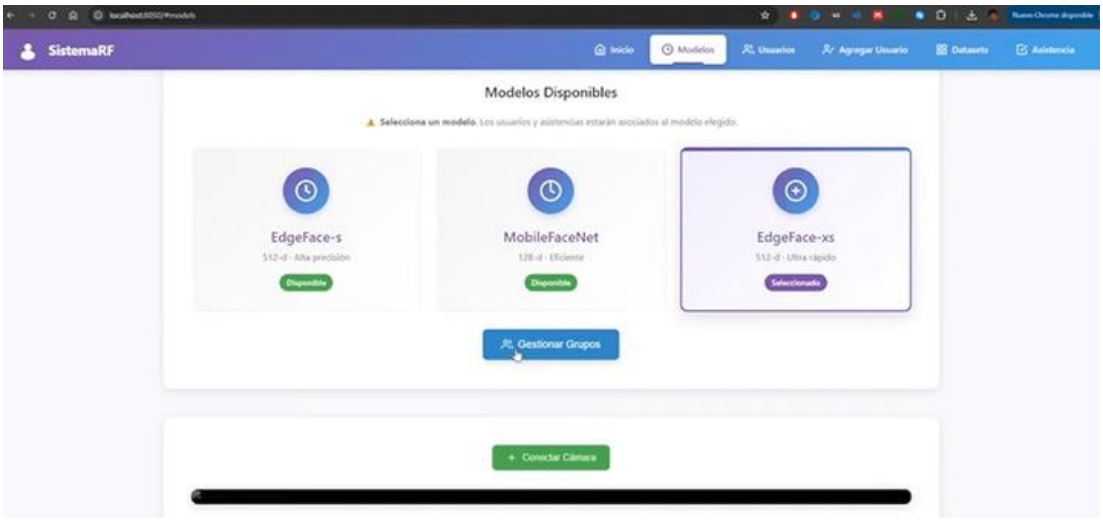
Validación del registro de usuario



- Mensaje de “Modelo no seleccionado”, lo que indica que no se escogió el modelo con el que realizará la prueba de este nuevo usuario e indica ir a la página de inicio para seleccionar modelo, antes de continuar con el registro de usuario.
- Para ello, seleccionó la opción “Modelos” del menú principal.

Figura 11

Selección de modelo a probar

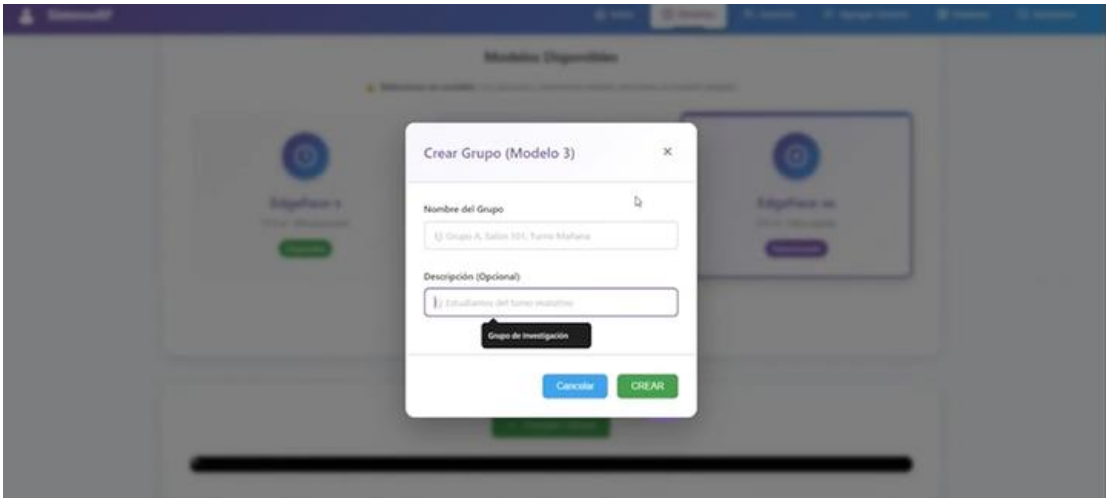


- Se seleccionó uno de los modelos, de los cuales serían utilizados para la realización de las pruebas.

- Y se creó y añadió un grupo en el que va a estar almacenado el usuario, con el botón “Gestionar grupos”

Figura 12

Creación de grupo



- Se completó el formulario con los campos “nombre de Grupo” y “Descripción”, creando así el grupo.
- Realizado ello, retornamos a Figura 04.

Figura 13

Formulario por completar para la validación de registro de usuario

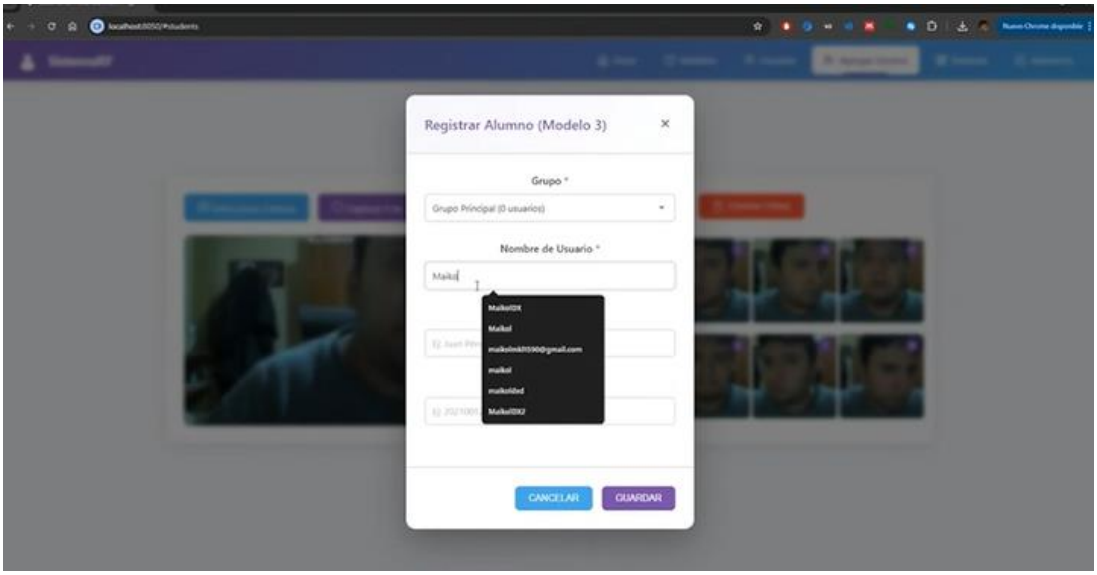
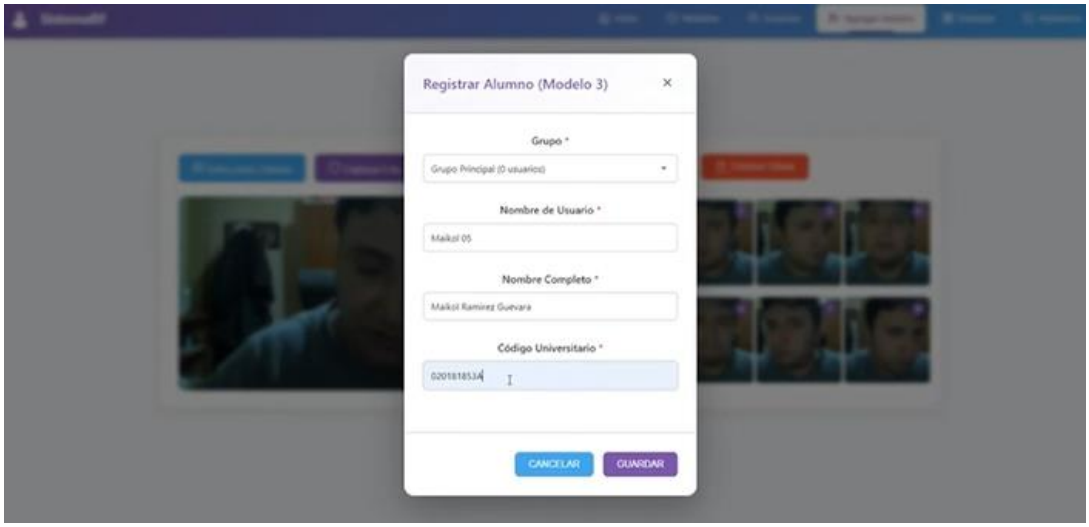


Figura 14

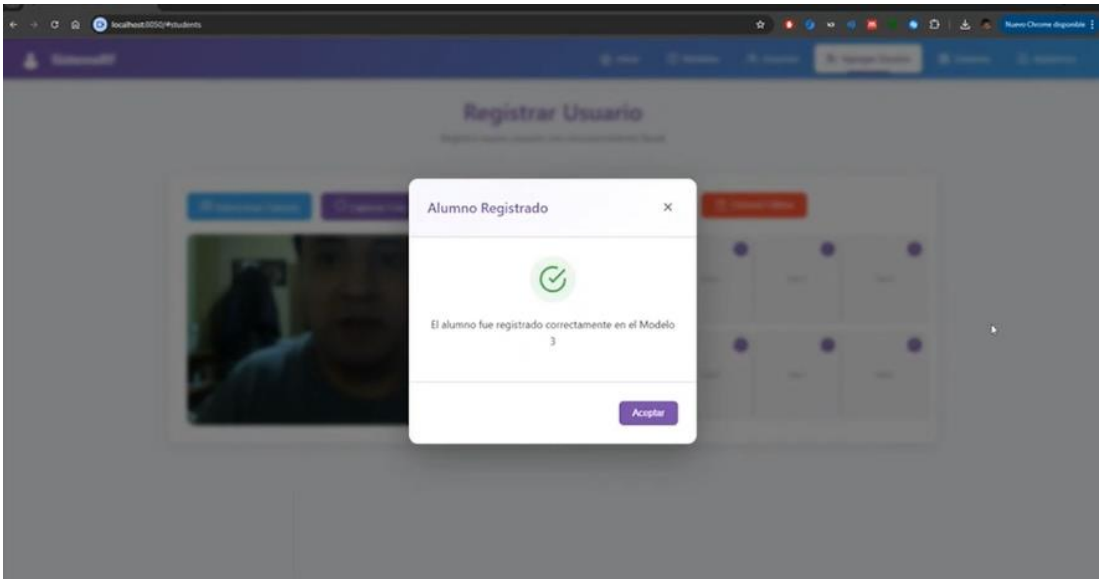
Formulario completado para la validación de registro de usuario



- Se completó el formulario donde solicita el Grupo, Nombre de Usuario, Nombre Completo y Código, completados los campos se realizó el guardado.

Figura 15

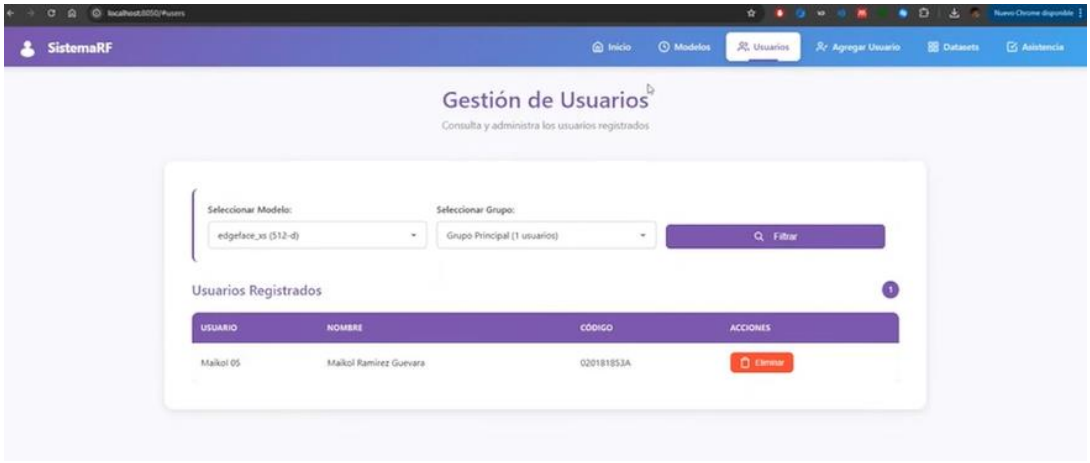
Validación exitosa del registro de usuario



- Mensaje emergente de “El usuario fue registrado correctamente en el Modelo”, siendo un indicativo que el registro fue exitoso

Figura 16

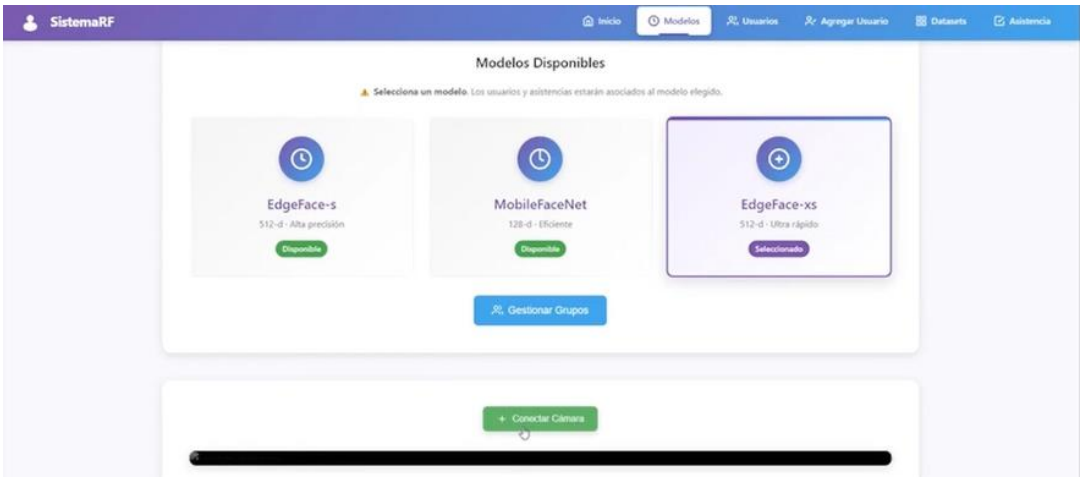
Comprobación del registro



- Se comprobó el registro en la ventana “Gestión de Usuarios”, con los campos completados por Modelo y Grupo.

Figura 17

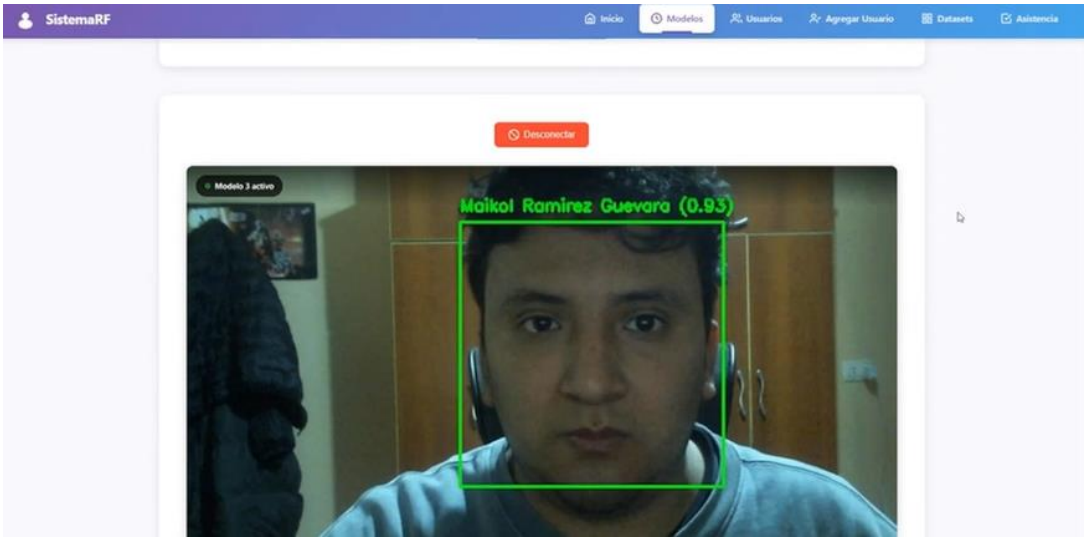
Selección de cámara para prueba del modelo con registro de usuario



- Se conectó a la cámara, previamente enlazada a la red, con la opción “Seleccionar cámara”.

Figura 18

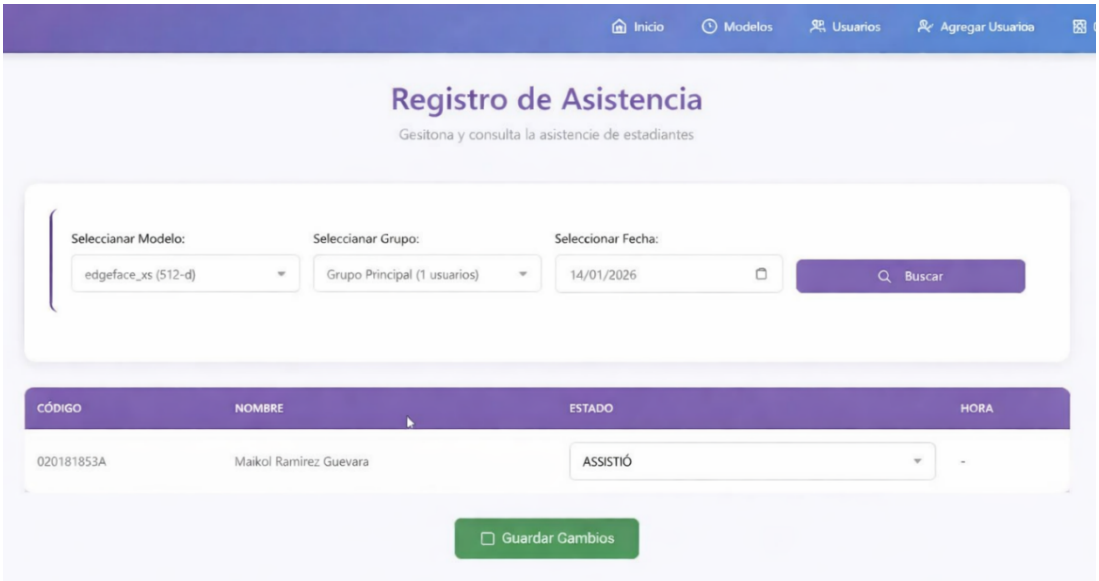
Prueba del modelo con registro de usuario



- El modelo, indicó una precisión del 0.93, con los factores de brillo y luz en el ambiente en el que el usuario está ubicado.

Figura 19

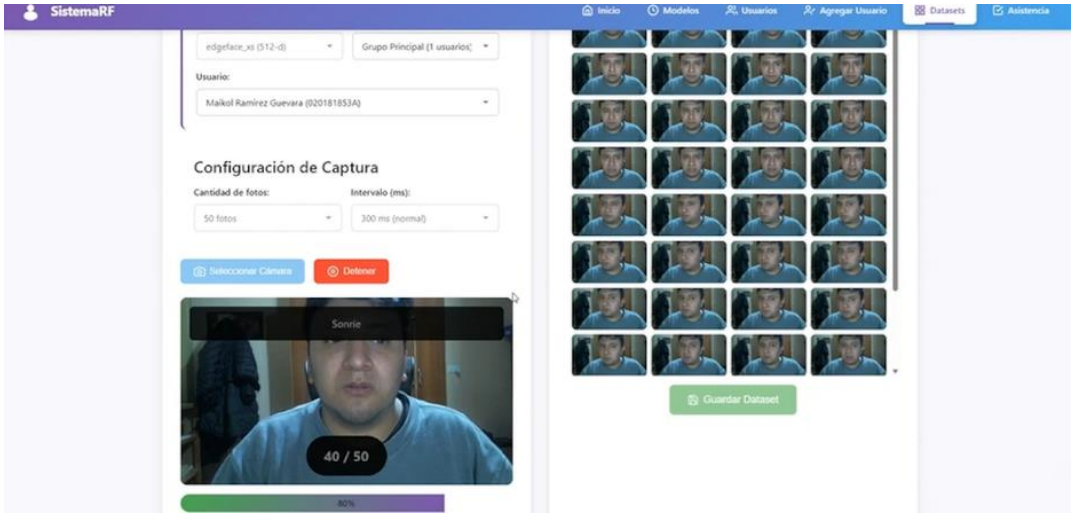
Asistencia de usuario por modelo



- Se comprobó que el usuario “Maikol Ramirez Guevara” fue registrado con el estado de “Asistió” en el Registro de Asistencia mediante el reconocimiento facial

Figura 20

Captura inicial de los Datasets para la comparativa de modelos



- Se seleccionó al usuario, y se configuró la captura de imágenes con la cantidad de fotos e intervalo (ms)
- Se conectó a la cámara, iniciando la captura de datasets.

Figura 21

Captura intermedia de los Datasets para la comparativa de modelos

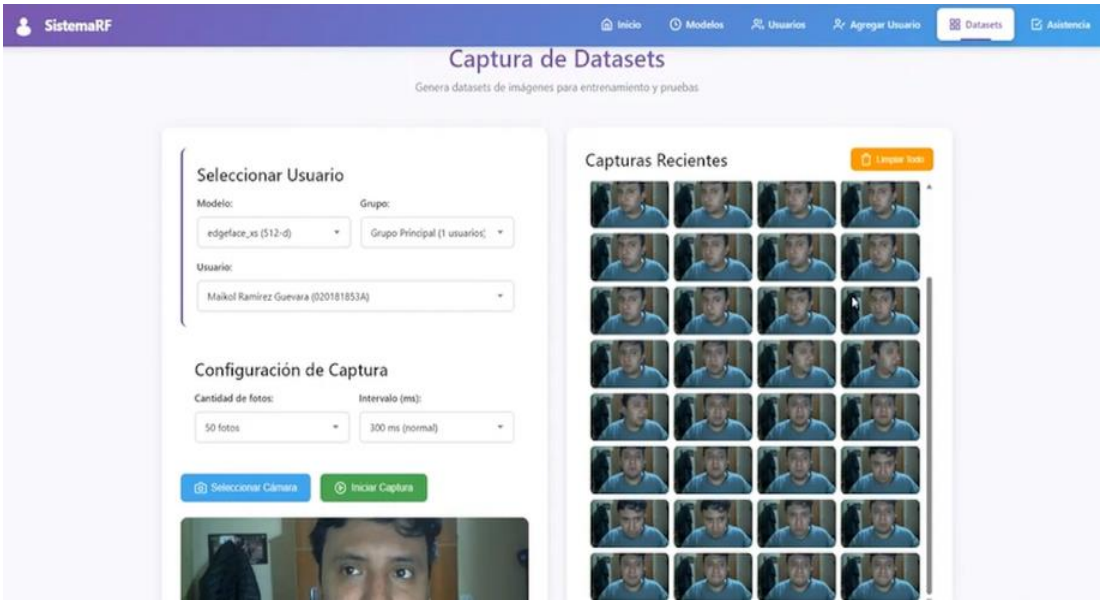
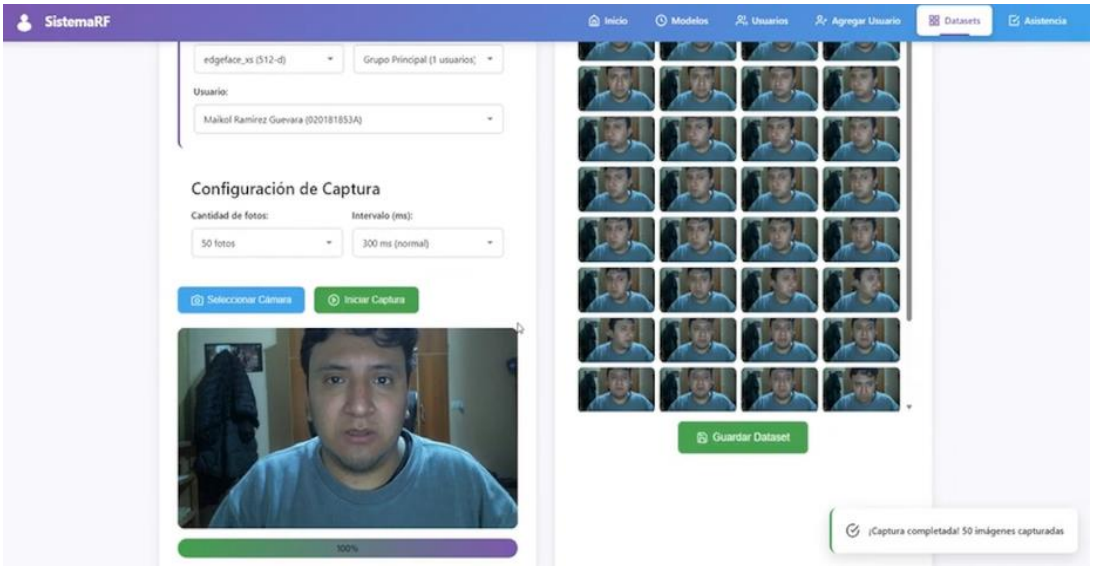


Figura 22

Captura completa de los Datasets para la comparativa de modelos



- Se guardó mediante la opción “Guardar Dataset”, completando así los

5.2.5. Selección del modelo final de reconocimiento facial.

Para evaluar de forma comparativa los tres modelos de reconocimiento facial (EdgeFace-S, EdgeFace-XS y MobileFaceNet) bajo condiciones equivalentes, se construyó un conjunto de prueba a partir del dataset de imágenes faciales capturadas por usuario a través de la interfaz web del sistema. Del total de imágenes disponibles por usuario, 8 ya se habían destinado al registro en el dispositivo (generación del embedding); las imágenes restantes constituyeron el conjunto reservado para la evaluación de modelos.

De este conjunto reservado se seleccionaron aleatoriamente imágenes pertenecientes a los 10 usuarios registrados, lo que representa aproximadamente el 30% de las imágenes disponibles para evaluación. Esta muestra conformó el conjunto de prueba común para los tres modelos evaluados, lo que garantiza una comparación justa al someter a EdgeFace-S, EdgeFace-XS y MobileFaceNet exactamente a las mismas pruebas.

Las imágenes seleccionadas no se procesaron a través de la interfaz de registro de asistencia del dispositivo, sino que se entregaron directamente a cada modelo, con el fin de

aislar el desempeño intrínseco del modelo de variables externas (latencia de hardware, captura en tiempo real, condiciones de iluminación en vivo).

Cada prueba consistió en una comparación 1:1 entre la imagen de prueba y el embedding facial de un usuario almacenado en la base de datos. El modelo respondió a la pregunta "¿Esta imagen corresponde al usuario consultado?".

Matrices de confusión

A continuación, se describe la interpretación de los componentes de la matriz de confusión en la identificación de usuarios.

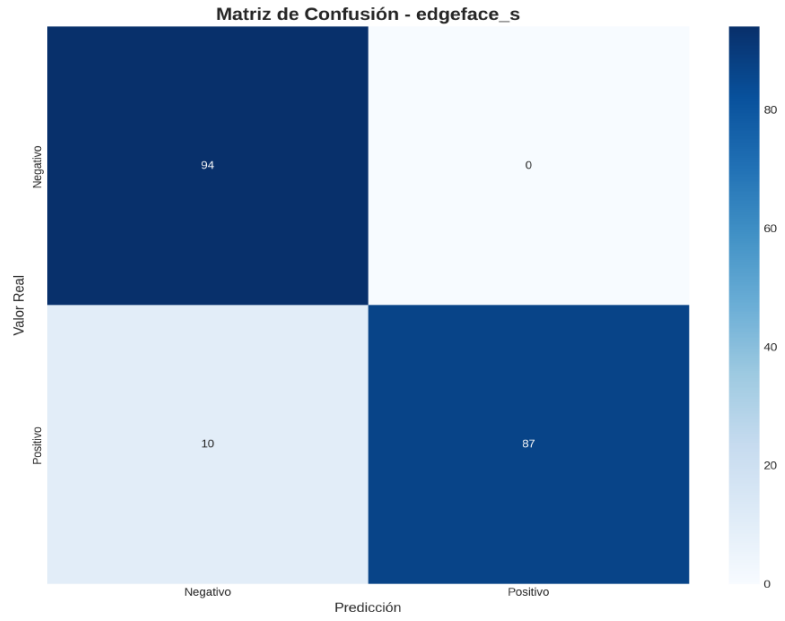
- **Verdaderos Positivos (TP):** Usuario identificado correctamente
- **Falsos Positivos (FP):** Usuario A identificado incorrectamente como Usuario B
- **Falsos Negativos (FN):** No se logra identificar al usuario correcto
- **Verdaderos Negativos (TN):** Se determina correctamente que el rostro no pertenece al usuario consultado
- **Métrica crítica:** FAR (False Acceptance Rate)

En identificación de usuarios, FAR mide la probabilidad de confundir un usuario con otro. FAR = 0% significa que el modelo NUNCA identifica al Usuario A como Usuario B.

- **Modelo EdgeFace-S**

Figura 23

Matriz de confusión – edgeFace-s



Resultados: TN=94, FP=0, FN=10, TP=87

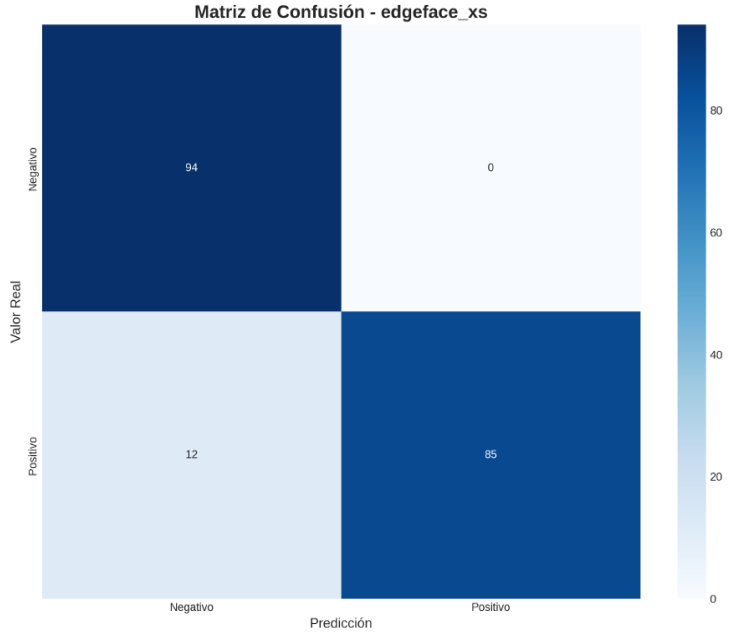
- FP = 0 (FAR = 0%): Nunca confunde usuarios entre sí. En 191 pruebas, jamás identificó Usuario A como Usuario B. Esta es la característica más importante para un sistema de identificación confiable.
- TP = 87, FN = 10: Identifica correctamente al usuario el 88.7% de las veces. Los falsos negativos ocurren principalmente bajo condiciones extremas (pose de perfil, oclusión severa).
- Precisión = 100%: Cuando dice "Este es el Usuario X", siempre acierta.
- Recall = 88.7%: De todas las imágenes del Usuario X, identifica correctamente 88.7%.

EdgeFace-S prioriza evitar confusiones de identidad (FP=0) manteniendo alta tasa de reconocimiento. En un dataset de 50 usuarios, esto significa 0 confusiones vs 6 usuarios no reconocidos en primera instancia (requerirán reintento).

- **EdgeFace-XS**

Figura 24

Matriz de confusión – edgeFace-xs



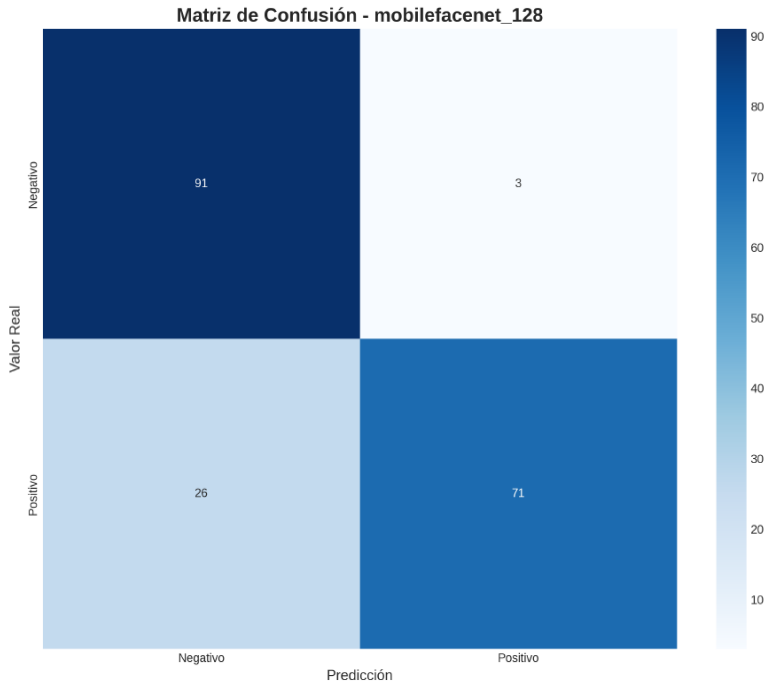
Resultados: TN=94, FP=0, FN=12, TP=85

- EdgeFace-XS mantiene la característica crítica de FAR=0% (nunca confunde usuarios) con rendimiento casi idéntico a EdgeFace-S. La diferencia de solo 1 caso adicional de FN lo hace excelente opción cuando los recursos computacionales son limitados, pero se requiere mantener cero confusiones entre usuarios.

- **MobileFaceNet**

Figura 25

Matriz de confusión – mobileFaceNet



Resultados: TN=91, FP=3, FN=23, TP=74

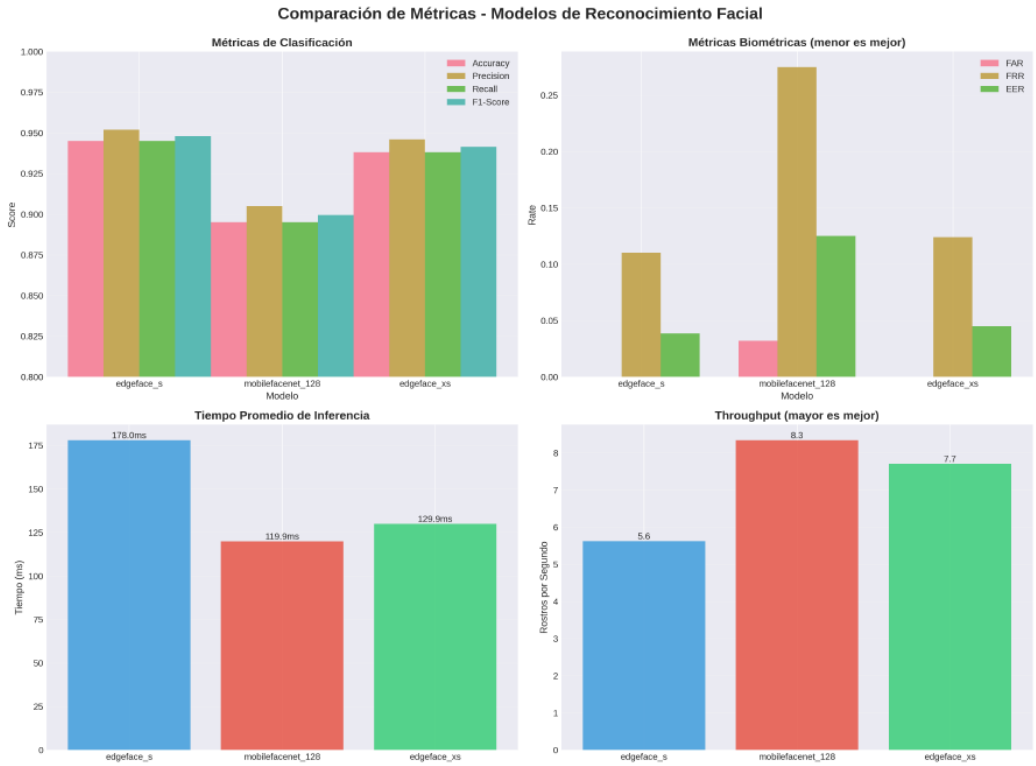
- FP = 3 (FAR = 3.2%): El modelo confunde usuarios entre sí. En 3 casos de 94, identificó incorrectamente Usuario A como Usuario B.
- FN = 23 (FRR = 27.5%): Alta tasa de fallo al identificar usuarios legítimos.

Embeddings de 128 dimensiones insuficientes para distinguir usuarios con características faciales similares. En un dataset de 30 usuarios con 300 consultas, MobileFaceNet-128 produciría ~9-10 confusiones de identidad. Esto lo descalifica para aplicaciones donde la identidad correcta del usuario es crítica.

Comparación de métricas

Figura 26

Comparación de métricas – Modelos de reconocimiento Facial



Se comparan cuatro dimensiones del rendimiento en la identificación de usuarios:

Tabla 7

Tabla comparación de métricas

Métrica	EdgeFace-S	EdgeFace-XS	MobileFaceNet
Accurracy	94.50%	93.80%	89.50%
Precision	95.20%	94.60%	90.50%
Recall	94.50%	93.80%	89.50%
F1-Score	94.80%	94.15%	89.95%

Nota. Elaboración propia.

- Accuracy 94.5%: EdgeFace-S logra identificar correctamente al usuario en 94.5% de cada 100 consultas. Hay una diferencia de 5% con MobileFaceNet-128 que se podría decir son 50 identificaciones correctas adicionales.
- Precision 95.2%: Cuando EdgeFace-S dice "Este es el Usuario X", acierta 95.2% de las veces. Alta precisión minimiza confusiones entre usuarios del dataset.

- Recall 94.5%: De todas las imágenes del Usuario X, EdgeFace-S identifica correctamente 94.5%. Esto indica robustez a variaciones del mismo usuario (diferentes expresiones, poses, iluminación).

Tabla 8

Tabla de métricas biométricas

Métrica Biométrica	EdgeFace-S	EdgeFace-XS	MobileFaceNet
FAR	0.00%	0.00%	3.20%
FRR	11.00%	12.40%	27.50%
EER	3.85%	4.50%	12.50%

Nota. Elaboración propia.

- FAR = 0% (Crítico): EdgeFace-S y EdgeFace-XS NUNCA confunden usuarios entre sí. MobileFaceNet-128 con FAR=3.2% confundirá ~32 usuarios por cada 1000 identificaciones de usuarios diferentes. Esta diferencia descalifica MobileFaceNet-128 para identificación precisa de usuarios.
- EER = 3.85%: Indicador del balance óptimo entre no confundir usuarios y reconocerlos consistentemente. EdgeFace-S logra EER en rango de sistemas premium (< 5%).

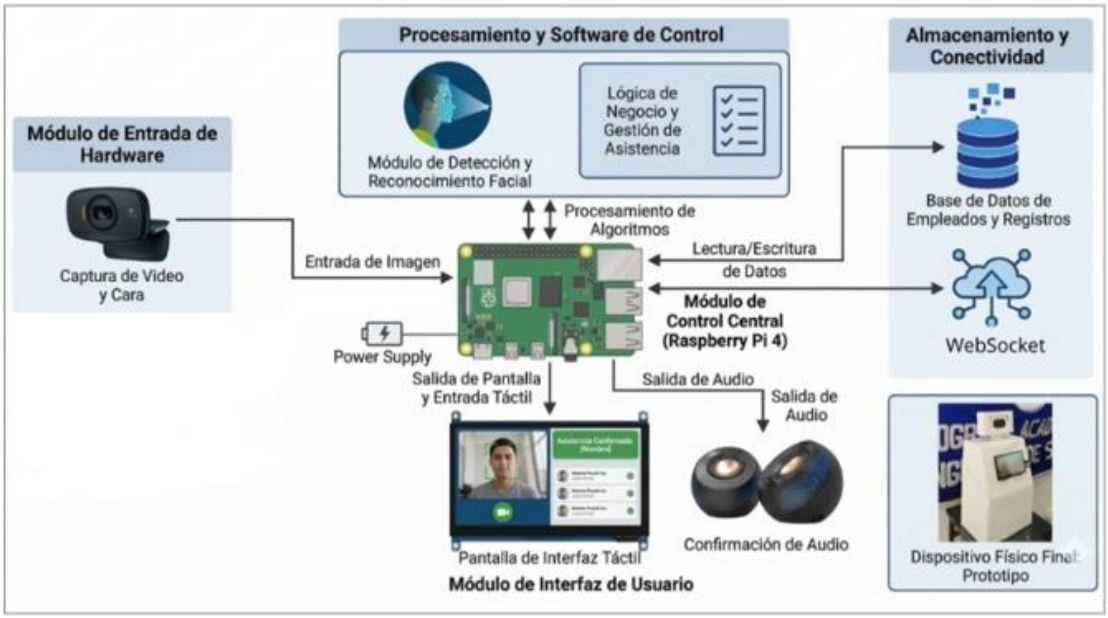
Selección final del modelo

Considerando los resultados mostrados anteriormente, EdgeFace-S demostró una superioridad en la identificación de usuarios con respecto a los otros modelos, con un 94.50% accurracy, 0% FAR, demostró ser un modelo óptimo frente a los otros. Además, se consideró que los modelos Edge de 512D superan ampliamente al modelo MobileFaceNet de solo 128d, por lo que, en la práctica, los modelos Edge tendrán mejores resultados. Por lo tanto, se eligió el modelo EdgeFace-S como el óptimo para este caso de estudio.

5.3 Resultado 3: Diseño de la arquitectura del dispositivo inteligente de registro de asistencia

Figura 27

Arquitectura del dispositivo



El diseño de la arquitectura del dispositivo integra componentes de hardware y software trabajando en conjunto. En las sesiones anteriores ya se mencionó la arquitectura del software, por lo que ahora se explica la parte del hardware.

Arquitectura de Hardware

- **Componente Central - Raspberry Pi 4B:**

  El Raspberry Pi 4B es el núcleo de todo el proyecto, por sus especificaciones técnicas:

  - Procesador: Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz

  - Memoria RAM: 4GB LPDDR4-3200 SDRAM

  - Conectividad: Wi-Fi 802.11ac, Bluetooth 5.0, Gigabit Ethernet

  - Puertos: 2 USB 3.0, 2 USB 2.0, 2 micro-HDMI

  - GPIO: 40 pines para conexiones externas

- **Sistema de Captura - Cámara:**

  Se usa una cámara web normal

  - Sensor: 8 megapíxeles

  - Resolución de video: 1080p a 30 FPS, 720p a 60 FPS

- **Sistema de Visualización - Pantalla LCD Táctil:**

  Se integró una pantalla táctil de 7 pulgadas:

  - Resolución: 800×480 píxeles

  - Interfaz táctil

  - Conexión: Mediante HDMI

- **Sistema Operativo:**

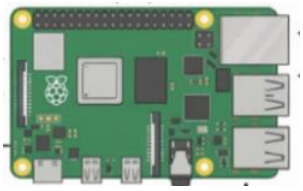
  Raspberry Pi OS (64-bit) basado en Debian, optimizado para el hardware del Raspberry Pi y compatible con todas las librerías necesarias.

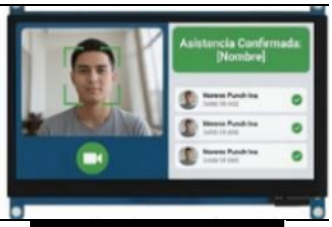
  Este diseño nos proporciona un equilibrio entre costo, eficiencia y facilidad de mantenimiento a futuro, completando las fases de desarrollo del dispositivo.

A continuación, se presenta la tabla resumen de los componentes que conforman la arquitectura del dispositivo inteligente desarrollado.

Tabla 9.

Componentes del dispositivo inteligente de registro de asistencia

Componente	Descripción de funcionamiento dentro del sistema	Imagen
Raspberry Pi 4B	Actúa como la unidad central de procesamiento del sistema, encargándose de ejecutar los algoritmos de reconocimiento facial, gestionar la lógica de negocio del registro de asistencia y coordinar la comunicación entre los diferentes módulos de hardware y software.	
Cámara	Dispositivo encargado de la captura de video en tiempo real, permitiendo la adquisición de imágenes faciales de los usuarios para su posterior procesamiento, detección y reconocimiento.	

Pantalla LCD táctil de 7”	Permite la interacción directa con el usuario mediante una interfaz gráfica, mostrando información en tiempo real como el estado del sistema, resultados del reconocimiento y opciones de gestión.	
Fuente de alimentación	Proporciona la energía eléctrica necesaria para el funcionamiento continuo del Raspberry Pi y los dispositivos periféricos conectados, garantizando la estabilidad del sistema.	
Base de datos SQLite	Sistema de almacenamiento embebido encargado de gestionar la información de usuarios, modelos y registros de asistencia, permitiendo operaciones eficientes de lectura y escritura de datos.	

Nota. Elaboración propia.

5.4 Resultado 4: Construcción del dispositivo inteligente

La construcción del dispositivo inteligente se desarrolló a partir de la integración ordenada de componentes electrónicos y de procesamiento, con el propósito de implementar un sistema de reconocimiento facial funcional, confiable y de fácil interacción con el usuario. El núcleo del sistema es la Raspberry Pi, la cual actúa como unidad central de control y procesamiento, permitiendo la ejecución de algoritmos de visión por computadora y reconocimiento facial en tiempo real.

El dispositivo incorpora una cámara web HD, ubicada en la parte frontal, cuya función es capturar imágenes del rostro del usuario bajo condiciones normales de iluminación y ambiente; dichas imágenes son enviadas a la Raspberry Pi, donde se realiza el procesamiento necesario para la detección facial y la posterior comparación con los registros almacenados en el sistema.

Asimismo, se integró una pantalla táctil de pequeño formato, la cual cumple la función de interfaz hombre máquina. A través de esta pantalla, el usuario recibe

información visual clara y comprensible, como mensajes de bienvenida, instrucciones del proceso y el resultado del reconocimiento facial, facilitando así una experiencia de uso sencilla y eficiente.

5.4.1. Diseño del dispositivo inteligente.

La Figura 28 muestra el diseño tridimensional (3D) del dispositivo inteligente, el cual fue desarrollado previamente con el fin de definir la distribución física de los componentes, la ergonomía del sistema y la estética general del dispositivo final. Este diseño 3D permitió planificar de manera adecuada el ensamblaje del hardware, considerando aspectos como la ubicación de la cámara, la visibilidad de la pantalla y la protección de los componentes internos mediante una carcasa compacta.

En conjunto, la construcción del dispositivo integra de forma eficiente hardware y software, obteniendo como resultado un sistema autónomo orientado a aplicaciones de seguridad, control de acceso y automatización, respaldado por un diseño estructural previamente modelado en 3D.

Figura 28

Diseño prototipo 3D del dispositivo inteligente

Nota. Elaboración propia.

Previamente se mencionó la arquitectura del software y hardware, con lo que se conllevo a la construcción del dispositivo.

5.4.2. Construcción del dispositivo inteligente.

- **Parte externa:**

En la Figura 29 se presenta la construcción de la parte superior del dispositivo, en la cual se integró el sistema de captura mediante la instalación de la cámara en una estructura diseñada para garantizar su estabilidad y correcta orientación, permitiendo así la adecuada adquisición de imágenes faciales durante el proceso de reconocimiento.

Figura 29

Resultado de la parte superior

Figura 30

Resultado de la parte inferior sin pantalla táctil en posición

En la Figura 30 se muestra la parte inferior del dispositivo en una fase inicial, donde aún no se ha incorporado la pantalla táctil, evidenciándose la estructura base destinada a albergar los componentes electrónicos y facilitar su organización física dentro del prototipo.

Posteriormente, en la Figura 31 se observa la integración de la pantalla táctil en la parte inferior del dispositivo, la cual fue instalada en una posición accesible para el usuario, permitiendo la interacción directa con el sistema a través de la interfaz gráfica desarrollada.

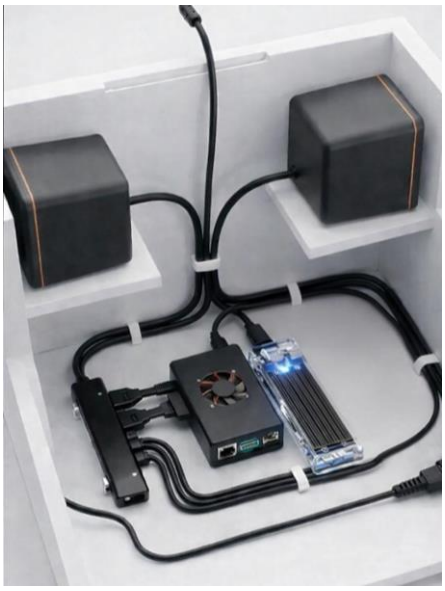
Figura 31

Resultado de la parte inferior con pantalla táctil en posición

- **Parte interna:**

Figura 32

Resultado de la parte interna que compone el dispositivo



En la Figura 32 se detalla la disposición interna del dispositivo, donde se ubican los componentes principales como la unidad de procesamiento, los módulos de alimentación y los dispositivos de conexión, organizados de manera que se optimice el espacio y se garantice un adecuado funcionamiento del sistema.

- **Resultado final del dispositivo inteligente:**

Finalmente, en la Figura 33 se presenta el prototipo completamente ensamblado, en el cual se integran todos los componentes tanto internos como externos, evidenciando la estructura final del dispositivo inteligente de registro de asistencia, listo para su funcionamiento en un entorno real.

Figura 33

Resultado del dispositivo

Como parte del proceso, el dispositivo se instaló en GICDIAC para las pruebas consiguientes.

5.5 Resultado 5: Evaluar el rendimiento del dispositivo inteligente en el registro de asistencia.

Para las pruebas requeridas, se brindó una lista de usuarios para el registro de manera manual, el cual comprobó a la efectividad del registro.

Figura 34

Dispositivo instalado en GICDIAC

- ***Pruebas de como el dispositivo registró asistencia de usuarios:***

Figura 35

Ejemplo de captura de prueba

Figura 36

Ejemplo de captura de prueba

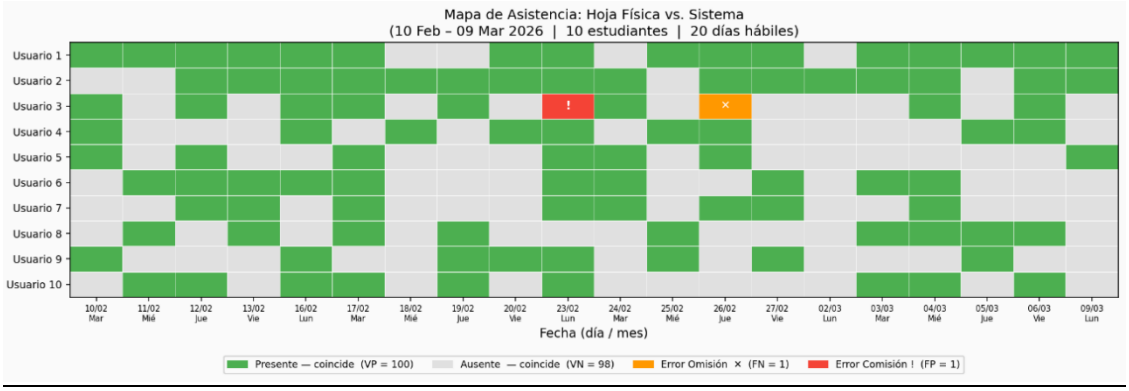
En la figura 34 y 35 se muestra un ejemplo de cómo los usuarios hicieron uso del sistema para registrar su asistencia.

Para evaluar el rendimiento del dispositivo, se consideró que, para evaluar el sistema con 10 usuarios registrados, se debía tener un doble control de asistencia, es decir, registrar las asistencias usando el dispositivo y a la vez registrando de manera tradicional, mediante una hoja de asistencia tradicional, de tal modo de que al término de los 20 días de pruebas se puedan cotejar los resultados y demostrar si el dispositivo era óptimo o no.

Es tal que al concluir las pruebas se obtuvieron los siguientes resultados:

Figura 37

Mapa de Asistencia: Hoja Física vs Sistema



Se obtuvo un mapa donde cada celda representa la intersección entre usuario y día hábil, el color indica si es que hay coincidencia entre la hoja física y el dispositivo. Los resultados obtenidos son:

Tabla 10

Tabla de resultados generales

Tipo	Registros	Porcentaje
VP - Aciertos de presencia	100	50%
VN - Aciertos de ausencia	98	49%
FN - Falsos negativos	1	0.5%
FP - Falsos positivos	1	0.5%
Total	200	100%

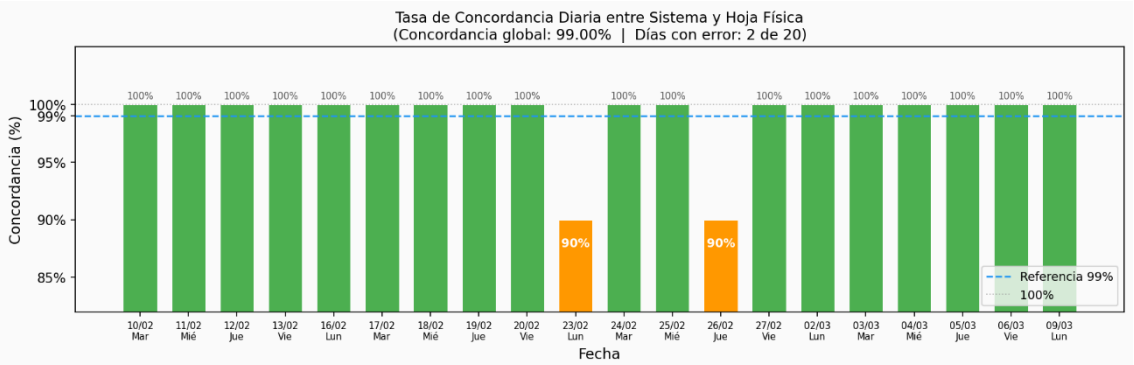
Nota. Elaboración propia.

En estas pruebas realizadas se identificaron 2 errores:

- FN – 26/02 - Usuario 3: Asistió y marcó en la hoja, pero en el sistema no se encontró el registro de ese día.
- FP – 23/02 – Usuario 3: No se contempló su registro en la hoja, pero sí apareció su registro en el sistema.

Figura 38

Tasa de concordancia diaria



Se obtuvo un gráfico de barras calculando la exactitud del día a día sobre los 10 usuarios. Es la Exactitud (Acurracy) aplicada a cada día en lugar del total acumulado.

Tabla 11

Tabla de resultados por día

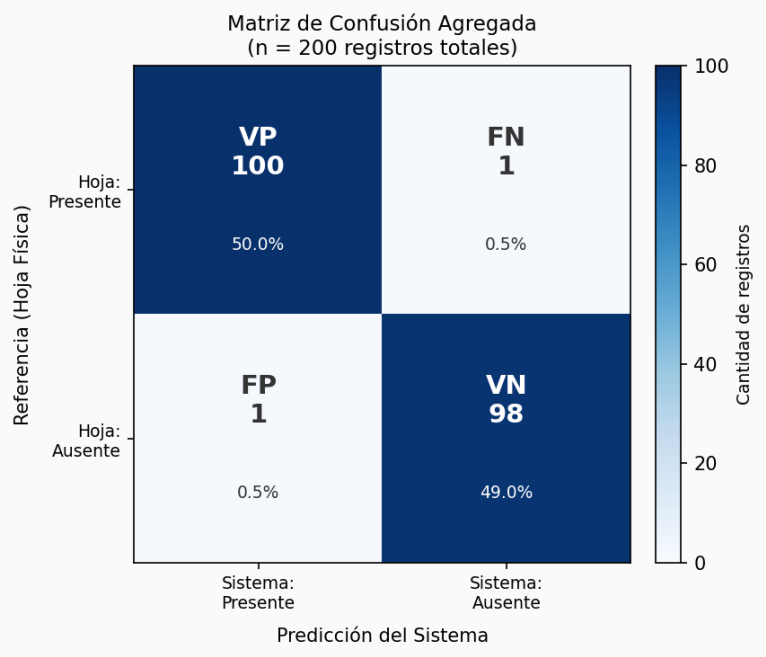
Tipo de barra	Días	Porcentaje
Verdes (sin error)	18 de 20	100%
Anaranjadas (con error)	2 de 20	90%

Nota. Elaboración propia.

La línea azul punteada es un umbral en referencia al 99% de exactitud global, este criterio se estableció como lo aceptable.

Figura 39

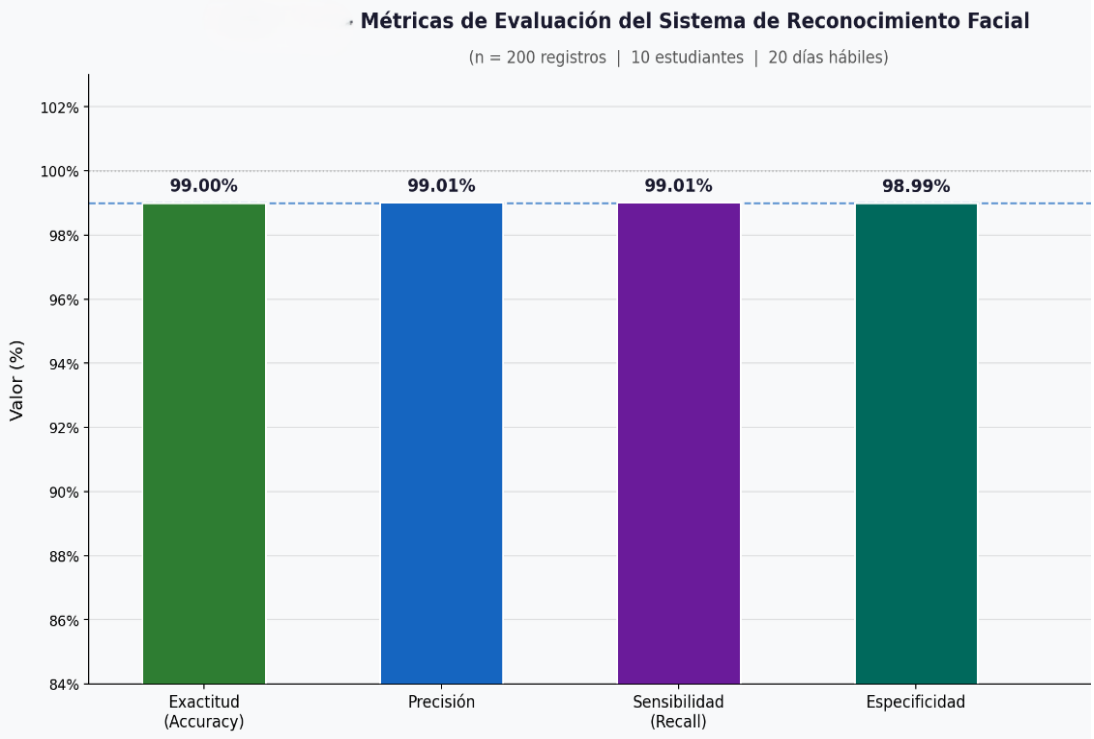
Matriz de confusión



Esta tabla resume todos los aciertos y errores registrados, todos los valores de la matriz son la base para obtener todas las métricas de rendimiento (Figura 40).

Figura 40

Métricas de evaluación del dispositivo



Se graficaron los cuatro indicadores de rendimiento del dispositivo, en la que se evidencia que el dispositivo tiene una exactitud de 99.00%, una precisión de 99.01%, una sensibilidad de 99.01%, y un 98.99% de especificidad.

6. DISCUSIÓN DE RESULTADOS

Los resultados evidencian que se tiene un dispositivo viable técnicamente, puesto que se pudo completar el desarrollo inicial del dispositivo de registro de asistencia mediante reconocimiento facial.

Para este estudio, se seleccionaron como mejores herramientas a una placa Raspberry Pi 4B, gestor de base de datos SQLite, lenguaje de programación python, y la evaluación de 3 modelos de reconocimiento facial: edgeFace-s, edgeFace-xs, mobilefaceNet. Estos resultados, coinciden con Kolekar y Patil (2023) junto con Bin-Hamed y Fatnassi (2022), en cuanto al uso del lenguaje de programación Python y la implementación de sistemas de reconocimiento facial en contextos de control de asistencia, lo cual se debe a la amplia disponibilidad de librerías especializadas en visión por computadora y aprendizaje automático que facilitan su integración en este tipo de soluciones, asimismo, se coincide con Bin-Hamed y Fatnassi (2022) en la selección de la Raspberry Pi como unidad central de procesamiento, debido a su bajo costo, portabilidad y capacidad suficiente para ejecutar algoritmos de reconocimiento en tiempo real.

Así mismo, para desarrollar el software se definió los requerimientos del sistema (funcionales y no funcionales), seguido del diagrama entidad relación, para posterior a ello realizar la programación del software, el cual estuvo divido en 3 módulos: backend, Interfaz visual para el usuario, y un módulo para la elección del mejor modelo de reconocimiento facial. Finalmente, con las interfaces ya programadas se pusieron a prueba

los 3 modelos de reconocimiento fácil, y de acuerdo a las métricas se seleccionó como mejor modelo a EdgeFace-S, pues obtuvo un 94.50% de accurracy, 0% de FAR.

Estos resultados presentan coincidencias con Azmi et al. (2023) en la implementación de un sistema de reconocimiento facial aplicado al control de asistencia y en la obtención de niveles de precisión cercanos, ya que en ambos casos se alcanzan valores alrededor del 94%, lo cual se explica por la adecuada selección de técnicas y el procesamiento de imágenes en condiciones reales, aunque difieren en los métodos utilizados, dado que Azmi et al. emplean algoritmos tradicionales como Haar Cascade y LBPH, mientras que en la presente investigación se incorporan modelos más avanzados basados en aprendizaje profundo. Asimismo, los resultados guardan relación con lo planteado por George et al. (2024), quienes proponen la familia de modelos EdgeFace optimizados para dispositivos con recursos limitados, coincidiendo particularmente en la selección de EdgeFace-S como una alternativa eficiente, sin embargo, se observa una diferencia en los niveles de precisión reportados, ya que George et al. (2024) alcanzan valores superiores al 99% en entornos controlados como el conjunto de datos LFW, mientras que en este estudio se obtuvo un 94.50% de accuracy, lo cual puede atribuirse a la implementación en un entorno real con variaciones de iluminación, posición y calidad de captura, lo que introduce mayores desafíos al sistema, no obstante, el resultado obtenido valida la viabilidad del modelo seleccionado en condiciones prácticas, destacando su equilibrio entre precisión y eficiencia computacional en dispositivos embebidos.

En la construcción del dispositivo, se elaboró preliminarmente un diseño en 3d, y con ello se procedió con el armado de cada uno de los componentes, siendo estos: una placa Raspberry Pi 4B, una cámara, una pantalla LCD, cable de fuente de alimentación. Uno de los puntos importantes fue la ubicación frontal de la cámara que favoreció la captura del rostro del usuario, así como la pantalla táctil que otorgó una interacción clara durante el

proceso de reconocimiento facial. Estos resultados presentan similitudes con los trabajos desarrollados por Bin-Hamed y Fatnassi (2022) y Azmi et al. (2023), en cuanto al uso de la Raspberry Pi como unidad central de procesamiento y la integración de una cámara para la captura de imágenes en tiempo real, lo cual evidencia una tendencia común en el diseño de sistemas de reconocimiento facial orientados a entornos con recursos limitados, debido a su bajo costo y capacidad funcional, sin embargo, la presente investigación introduce una diferencia relevante al considerar un diseño estructural previo mediante modelado en 3D y una organización más definida de los componentes físicos, incluyendo la incorporación de una pantalla táctil que mejora la interacción con el usuario, aspecto no detallado en los estudios comparados.

Tras evaluar el rendimiento del dispositivo con 10 usuarios durante 20 días de prueba, se encontró que el dispositivo presenta una exactitud de 99.00%, una precisión de 99.01%, una sensibilidad de 99.01%, y un 98.99% de especificidad. Estos resultados evidencian un desempeño superior en comparación con estudios previos, como los desarrollados por Ullah et al. (2022), Al-Ghrairi (2022) y Archana et al. (2022), en los cuales se reportan niveles de precisión entre el 95% y 96%, mientras que en la presente investigación se alcanzan valores cercanos al 99% en métricas como exactitud, precisión, sensibilidad y especificidad, esta diferencia puede atribuirse a la integración de modelos de reconocimiento facial optimizados para entornos con recursos limitados, así como a la implementación de un sistema completo que combina adecuadamente hardware y software en condiciones controladas de operación, no obstante, es importante señalar que estudios como el de Ullah et al. (2022) utilizaron conjuntos de datos considerablemente más grandes y condiciones más variables, incluyendo cambios significativos de iluminación y ángulos, lo que incrementa la complejidad del reconocimiento y puede afectar el rendimiento, de manera similar, Al-Ghrairi (2022) y Archana et al. (2022) emplearon algoritmos

tradicionales y datasets específicos que, si bien permiten validar la eficacia de sus propuestas, no alcanzan el mismo nivel de optimización logrado en este estudio, en consecuencia, las diferencias observadas se deben principalmente al tipo de modelo utilizado, al tamaño y control del entorno de evaluación y al nivel de integración del sistema, lo que permite concluir que la solución propuesta ofrece un alto grado de precisión en contextos reales con condiciones operativas controladas, destacando su viabilidad para aplicaciones de registro de asistencia mediante reconocimiento facial.

7. CONCLUSIONES

Se llegó a identificar y seleccionar con éxito las herramientas óptimas para el desarrollo del dispositivo inteligente de reconocimiento facial. La elección del modelo óptimo termino siendo edgeFace-xs, por la obtención de mejores resultados en el desarrollo proporcionando un ambiente eficiente y preciso para el sistema a desarrollar.

Se desarrolló el software combinando 3 módulos, base de datos, backend y frontend. Implementando las funcionalidades adecuadas para el reconocimiento facial en tiempo real, el correcto registro de nuevos usuarios y los reportes generados.

Se diseñó la arquitectura del dispositivo en cuestión, logrando una compatibilidad con el software desarrollado, logrando la cohesión de hardware y software.

Los resultados obtenidos demuestran que el dispositivo de registro de asistencia alcanza un nivel de concordancia del 99% con la hoja física de asistencia tradicional a lo largo de 20 días hábiles de evaluación con 10 usuarios. Estos resultados posicionan al dispositivo dentro de los estándares requeridos para su implementación como alternativa viable al registro manual de asistencia.

8. RECOMENDACIONES

Se recomienda a futuros investigadores ampliar el tamaño y diversidad de la muestra utilizada durante la fase de evaluación, incorporando un mayor número de usuarios y condiciones variables de iluminación, ángulos y expresiones faciales, con el fin de validar la robustez del sistema en entornos más exigentes y mejorar la generalización del modelo de reconocimiento facial.

Se recomienda implementar mecanismos adicionales de seguridad, como autenticación multifactor o códigos QR, con el objetivo de reforzar la confiabilidad del sistema y prevenir posibles suplantaciones de identidad.

Se propone optimizar el consumo energético y el rendimiento del dispositivo mediante el uso de técnicas de procesamiento eficiente o hardware especializado, lo cual permitiría mejorar la autonomía del sistema y facilitar su implementación en contextos con limitaciones de energía o infraestructura.

Se sugiere considerar la integración del sistema con plataformas institucionales o sistemas de gestión académica o laboral, de manera que los registros de asistencia puedan ser sincronizados automáticamente, mejorando la eficiencia administrativa y ampliando la aplicabilidad del dispositivo en entornos reales.

9. REFERENCIAS

Aguileta, A., Brena, R., Molino, E., & Galván C. (2022). *Facial Expression Recognition from Multi-Perspective Visual Inputs and Soft Voting.* Sensors, 22(11), 4206. https://doi.org/10.3390/s22114206

Al-Ghrairi, A.; Mohammed, A. & Sameen, E. (2022). *Face detection and recognition with 180-degree rotation based on principal component analysis algorithm. IAES International Journal of Artificial Intelligence*. IJ, 11(2) https://doi.org/10.11591/ijai.v11.i2.pp593-602

Anyaypoma, J., & Hoyos , A. (2021). *Propuesta de mejora del proceso de control de asistencia del personal para optimizar la gestión administrativa en la unidad territorial Cajamarca del Programa Nacional de Apoyo Directo a los más pobres – Juntos.* Universidad Privada del Norte. Obtenido de https://hdl.handle.net/11537/9888

Archana, M.; Nitish, C. & Harikumar, S. (2022). *Real time Face Detection and Optimal Face Mapping for Online Classes.* Journal of Physics: Conference Series, 2161(1). https://doi.org/10.1088/1742-6596/2161/1/012063

Arias Gonzales, J., & Covinos, M. (2021). *Diseño y metodologia de la investigacion.* Enfoques Consulting EIRL.

Arispe Alburqueque, C., Yangali, J., Guerrero, M., Lozada, O., Acuña , L., & Areallano , C. (2020). *LA INVESTIGACIÓN CIENTÍFICA.* Guayaquil, Ecuador: Universidad Internacional de Ecuador.

Bin-Hamed, A., & Fatnassi, M. (2022). *Smart attendance monitoring system using facial recognition and Raspberry Pi.* Journal of Telecommunications and Digital Economy, 10(2), 45–60. https://doi.org/10.18080/jtde.v10n2.530

Chen, S., Liu, Y., Gao, X., & Han, Z. (2019). *MobileFaceNets: Efficient CNNs for Accurate Real-Time Face Verification on Mobile Devices*. arXiv. https://doi.org/10.48550/arXiv.1804.07573

Chen, Y., Wang, J., Chen, S., Shi, Z., & Cai, J. (2020). *Facial Motion Prior Networks for Facial Expression Recognition*. 2020 IEEE International Conference on Visual Communications and Image Processing, VCIP 2020. https://doi.org/10.1109/VCIP47243.2019.8965826

De La Cruz, J. (2023). *Automatización de procesos digitales y la atención al ciudadano en una municipalidad provincial de Lambayeque*. Repositorio UCV. https://hdl.handle.net/20.500.12692/124134

Espino, C. (2019). Sistema de información para el control de asistencia del personal de la empresa Global Sales Solutions Line Sucursal Perú https://doi.org/10.1016/J.VRIH.2020.04.002

George, A., Ecabert, C., Otroshi, H., Kotwal, K., & Marcel, S. (2024). *EdgeFace: Efficient Face Recognition Model for Edge Devices*. *IEEE Transactions on Biometrics Behavior and Identity Science*, 158–168. https://doi.org/10.1109/TBIOM.2024.3352164

Houssou, M.; Mahama, A.; Gouton, P. & Degla, G. (2022). *Robust Facial Recognition System using One Shot Multispectral Filter Array Acquisition System.* International Journal of Advanced Computer Science and Applications, 13(1), 25–33. https://doi.org/10.14569/IJACSA.2022.0130104

Hinojosa Mamani, J., Mamamani Gamarra, J., & Catacora Lucana, E. (2024). *PROYECTO DE TESIS: Guía práctica para investigación cuantitativa.* Sao Paulo, Brasil: EDITORA CIENTÍFICA DIGITAL LTDA. doi:10.37885/978-65-5360-556-5

Huang, Y., & Chen, H. (2020). *Face Recognition Under Low Illumination Via Deep Feature Reconstruction Network.* https://doi.org/10.1109/ICIP40778.2020.9191321

Irjanto, N. & Surantha, N. (2020). *Home Security System with Face Recognition based on Convolution Neur network.* https://dx.doi.org/10.14569/IJACSA.2020.0111152

Ishaq, K., & Bibi, S. (2023). IoT Based Smart Attendance System Using Rfid: A Systematic Literature Review. *arxiv*, 1-26. doi:doi.org/10.48550/arXiv.2308.02591

Jain, A., Ross, A., & Nandakumar, K. (2011). Introduction to Biometrics. En A. Jain, A. Ross, & K. Nandakumar, *Introduction to Biometrics* (pág. 312). Springer Science & Business Media. Obtenido de https://books.google.com.pe/books/about/Introduction_to_Biometrics.html?id=ZPt2xrZFtzkC&redir_esc=y

Ke, X., Lin, B., & Guo, W. (2022). *LocalFace: Learning significant local features for deep face recognition*. Image and Vision Computing, 123, 104484. https://doi.org/10.1016/j.imavis.2022.104484

Kolekar, S., & Patil, P. (2023). *A comprehensive study on artificial intelligence-based face recognition technologies. En IoT Based Control Networks and Intelligent Systems (Lecture Notes in Networks and Systems).* Springer. https://doi.org/10.1007/978-981-99-6586-1_17

Lau, E. (2022). *Gobierno digital para mejorar procesos de la calidad de información en docentes de una UGEL, Lambayeque*. Repositorio UCV. https://hdl.handle.net/20.500.12692/94237

Li, J., Ding, Y., Shao, Z., & Jiang, W. (2024). *Face recognition method based on fusion of improved MobileFaceNet and adaptive Gamma algorithm*. Journal of Franklin Institute, 361(17), 107306. https://doi.org/10.1016/j.jfranklin.2024.107306

López , E. (2017). *Raspberry Pi Fundamentos y Aplicaciones.* RA-MA S.A. Editorial y Publicaciones. Obtenido de https://www.google.com.pe/books/edition/Raspberry_Pi_Fundamentos_y_Aplicaciones/Zae6EAAAQBAJ?hl=es&gbpv=0

Lutz, M. (2009). Learning Python. En M. Lutz, *Learning Python* (pág. 1214). O'Reilly Media, Inc. Obtenido de https://books.google.com.pe/books/about/Learning_Python.html?id=4pgQfXQvekcC&redir_esc=y

Mondal, H., & Mondal, S. (2023). Methods of collecting and recording attendance of medical students in a classroom: A systematic review. *Journal of Education and Health Promotion, 12*(427), 1-8. doi:doi.org/10.4103/jehp.jehp_737_23

Mouafo, D., & Biaou, U. (2022). *Face recognition system for control access to restrictive domain. International Journal of Safety and Security Engineering.* (IJSSE), 12(2), 251–257. https://doi.org/10.18280/ijsse.120214

Ogla, R., Saeid, A., & Shaker, S. (2022). Technique for recognizing faces using a hybrid of moments and a local binary pattern histogram. *International Journal of Electrical and Computer Engineering (IJECE)*, 12(3), 2571-2581. https://doi.org/10.11591/ijece.v12i3.pp2571-2581

Salinas, C., Liu, A., & Sharaf, B. (2023). *Facial Morphometrics in Black Celebrities: Contemporary Facial Analysis Using an Artificial Intelligence Platform*.

Journal of Clinical Medicine, *12*(13), 4499. https://doi.org/10.3390/jcm12134499

Ullah, R., Hayat, H., Abid, A., Uzma A., Khan, J., Ullah, F., Hassan, S., Hasan, L., Albattah, W., Islami, M., & Karami, G. (2022). *A Real - Time Framework for Human Face Detection and Recognition in CCTV Images*. https://doi.org/10.1155/2022/3276704

Vahid, F., y Givargis, T. (2001). Embedded System Design: A Unified Hardware / Software Introduction. En F. Vahid, y T. Givargis, Embedded System Design: A Unified Hardware / Software Introduction (pág. 352). New York: John Wiley & Sons. https://books.google.com.pe/books/about/Embedded_System_Design.html?id=0ZW6QgAACAAJ&redir_esc=y

Vakili, M., Ghamsari, M., & Rezaei, M. (2020). Performance Analysis and Comparison of Machine and Deep Learning Algorithms for IoT Data Classification. *Cornell University*.

Warden, P., & Situnayake, D. (2019). inyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers. En P. Warden, & D. Situnayake, *inyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers* (pág. 504). O'Reilly Media, Inc. Obtenido de https://books.google.com.pe/books/about/TinyML.html?id=tn3EDwAAQBAJ&redir_esc=y

Yang, S.; Luo, P.; Loy, C. & Tang, X. (2018). *Faceness-Net: Face Detection through Deep Facial Part Responses*. IEEE Transactions on Pattern Analysis and

Machine Intelligence, 40(8), 1845–1859. https://doi.org/10.1109/TPAMI.2017.2738644

Yang, X. & Wei, Z. (2021). *Heterogeneous face detection based on multi-task cascaded convolutional neural network*. https://doi.org/10.1049/ipr2.12344

Zhang, H. & Chi, L. (2020). *End-to-End Spatial Transform Face Detection and Recognition*. Virtual Reality & Intelligent Hardware, 2(2), 119–131. https://doi.org/10.1016/J.VRIH.2020.04.002

Zhang, H. (2023). *Biometric Authentication and Correlation Analysis Based on Multimodal Biometrics*. https://doi.org/10.1155/2023/8389193

Zhu, Y. & Jiang, Y. (2020). *Optimization of face recognition algorithm based on deep learning multi feature fusion driven by big data*. Image and Vision Computing, 104, 104023. https://doi.org/10.1016/J.IMAVIS.2020.104023

10. ANEXOS

Anexo 1. *Código relacionado al módulo de registro de alumno*

```python
@app.post("/save/image_group/{group_id}", response_model=ImageResponse)
def save_base64_image_api(group_id:str, request_data:ImageRequest):
    print("Llegamos al método guardar")
    #Conectarse a la BD
    db_connector = SQLiteConnector("app_db.db")

    # guardar las imágenes
    for image_data in request_data.base64Images:
        print(image_data)

        db_connector.create_or_update(table_name=DbTable.BASE64_IMAGES,
                                      condition_dict={"id": str(uuid.uuid4())},
                                      data={"group_id": group_id,
                                            "name": request_data.name,
                                            "data": image_data})
    # Guardar comandos
    db_connector.create_or_update(table_name=DbTable.COMMANDS,
                                  condition_dict={"group_id": group_id},
                                  data={"command_name": request_data.command_name,
                                        "device_name": request_data.device_name})

    # Guardar datos del usuario
    db_connector.create_or_update(table_name=DbTable.USERS,
                                  condition_dict={"idusers": str(uuid.uuid4())},
                                  data={"group_id": group_id,
                                        "nombre_completo": request_data.user_fullname,
                                        "codigo_universitario": request_data.user_university_code})

    request_data_dict = request_data.model_dump()
```

Anexo 2. *Código relacionado al módulo de detección facial*

```python

def run_websocket_image_scan2():
    print("started run_websocket_image_scan 2")
    try:
        consecutive_frames = 0
        user_id_to_save = None

        while True:
            time.sleep(1)

            # if keyboard.is_pressed('q'):
            #     print("Saliendo del bucle...")
            #     break  # Sale del bucle si la tecla 'q' ha sido presionada

            with encodings_lock:
                # Capturar frame por frame
                video_capture = cv2.VideoCapture(0)
                # if video_capture.isOpened():
                #     print(f"La cámara se abrió con el índice 0")
                # else:
                #     print(f"No se pudo abrir la cámara con el índice 0")

                ret, frame = video_capture.read()

                resized_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)

                face_locations = face_recognition.face_locations(resized_frame)
                face_encodings = face_recognition.face_encodings(resized_frame, face_locations)

                video_capture.release()
                for (top, right, bottom, left), face_encoding in zip(face_locations, face_encodings):
                    matches = face_recognition.compare_faces(known_face_encodings, face_encoding)
                    name = ""
                    matched_ids_set = set()
                    for match_index, is_match in enumerate(matches):
                        if is_match:
                            match_group_id = known_face_groups[match_index]
                            if not match_group_id in matched_ids_set:
                                user_id_to_save = match_group_id
                                consecutive_frames += 1

                                if consecutive_frames >= 3:
                                    #CONTROLAR ASISTENCIA POR HORA----------------
                                    # Obtener la fecha actual
                                    fecha_actual = datetime.now()

                                    #Controlar las fechas de entrada: ASISTIÓ - TARDANZA - FALTO(En caso no llegue el alumno)
                                    # Formatear la fecha como una cadena con el formato "dd-mm-YYYY"
                                    fecha_formateada = fecha_actual.strftime("%d-%m-%Y")

                                    state = validState(fecha_actual)

                                    print("USUARIO POR REGISTRAR")
                                    name = known_face_names[match_index]
                                    set_control_command_for_match(match_group_id)
                                    save_asistencia(user_id_to_save, name, state, fecha_formateada)
                                    consecutive_frames = 0
                                matched_ids_set.add(match_group_id)
    except Exception as e:
        print("Exception:", e)

```

Anexo 3. *Código relacionado al módulo de registro de asistencia*

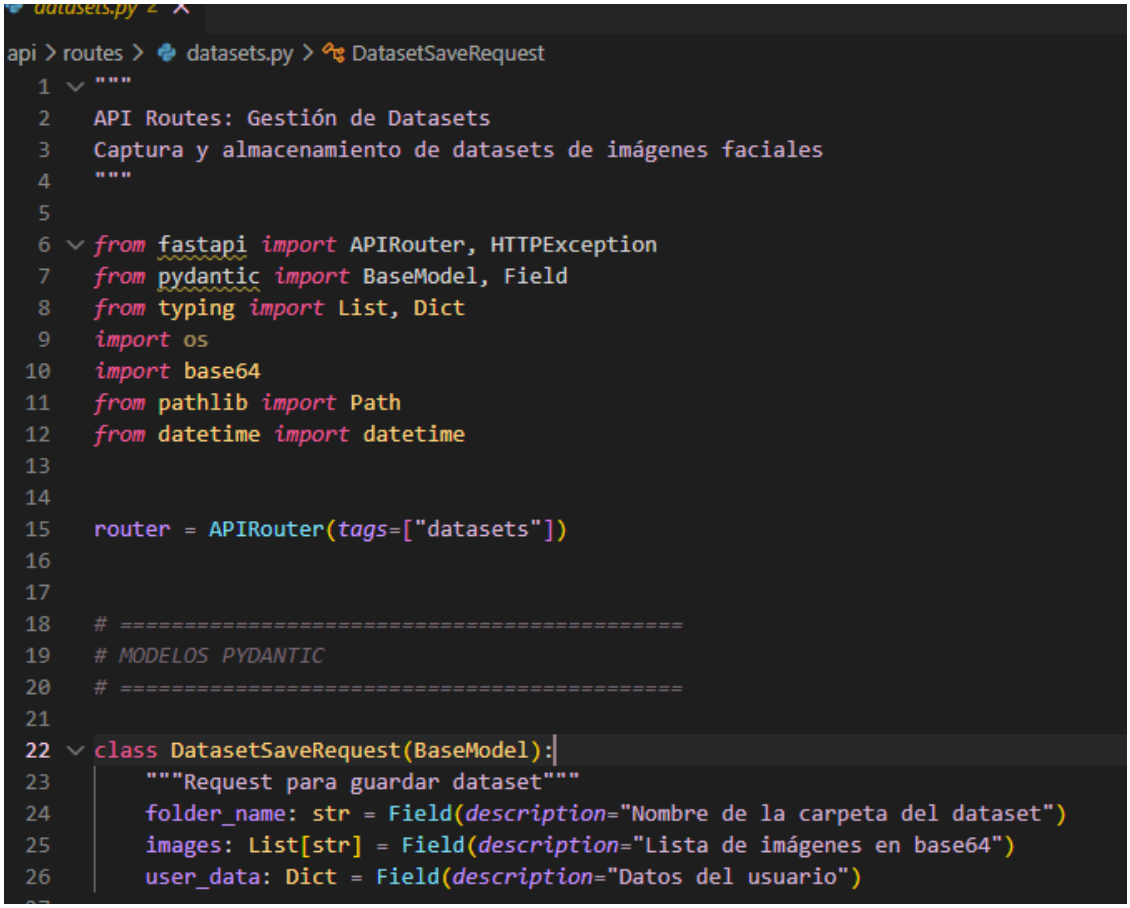
```python
def save_asistencia(id_usuario:str, name:str, state:str, fecha:str):
    try:
        # Obtener la fecha actual
        # fecha_actual = datetime.now()
        #Controlar las fechas de entrada: ASISTIO - TARDANZA - FALTO(En caso no llegue el alumno)
        # Conectar a la base de datos
        db_connector = SQLiteConnector("app_db.db")

        # Verificar si ya existe una asistencia para el usuario en la fecha actual
        existing_asistencia = db_connector.get_data(
            table_name=DbTable.ASISTENCIA,
            condition_dict={"users_idusers": id_usuario, "fecha": fecha}
        )

        # Si ya existe una asistencia para el usuario en la fecha actual, retornar un mensaje de error
        global message
        if existing_asistencia:
            #message = "presente"
            update_message("Alumno:" + name, "Ya Registrado!")
            start_lcd_thread()

            stop_lcd_thread()

            print("la asistencia para este usuario ya ha sido registrada hoy.")
            return "la asistencia para este usuario ya ha sido registrada hoy."

        # Si no existe una asistencia para el usuario en la fecha actual, guardar la nueva asistencia
        db_connector.create_or_update(
            table_name=DbTable.ASISTENCIA,
            condition_dict={"idasistencia": str(uuid.uuid4())},
            data={"tipo_asistencia": state,
                  "users_idusers": id_usuario,
                  "fecha": fecha}
        )

        #MOSTAR MENSAJE POR CONSOLA Y DISPLAY
        #message = "presente"

        print("Antes de entrar al display registro")

        update_message("Alumno:" + name, "Registrado!")
        start_lcd_thread()
        stop_lcd_thread()

        print("Asistencia guardada correctamente.")
        return "Asistencia guardada correctamente."
    except Exception as e:
        return f"Error al guardar la asistencia: {str(e)}"
```

Anexo 4. *Código relacionado a la gestión de asistencia*

```python
"""
API Routes: Gestión de Asistencias
Endpoints completos para consultar y guardar asistencias
"""

from fastapi import APIRouter, HTTPException
from pydantic import BaseModel, Field
from typing import List, Optional

from core.attendance_service import AttendanceService
from database.db_connector import SQLiteConnector
from database.db_models import DbTable


router = APIRouter(tags=["attendance"])

# Instancias globales (se inicializarán en main.py)
attendance_service: Optional[AttendanceService] = None
db_connector: Optional[SQLiteConnector] = None


# ============================================
# MODELOS PYDANTIC
# ============================================

class UsersResponse(BaseModel):
    """Response de asistencia de usuario"""
    id: str = Field(description="ID asistencia")
    user_university_code: str = Field(description="Código universitario")
    user_fullname: str = Field(description="Nombre completo")
    type_assintance: str = Field(description="Tipo de asistencia")
    hora: str = Field(description="Hora", default="")
    confianza: float = Field(description="Confianza", default=0.0)


class AttendanceRequest(BaseModel):
    """Request para actualizar asistencia"""
    user_university_code: str = Field(description="Código universitario")
    user_fullname: str = Field(description="Nombre completo")
    type_assintance: str = Field(description="Tipo de asistencia")


# ============================================
# ENDPOINTS
# ============================================

@router.get("/asistencia/{fecha}", response_model=List[UsersResponse])
```

```python
@router.get("/asistencia/{fecha}", response_model=List[UsersResponse])
async def get_attendance_by_date(fecha: str):
    """
    Obtiene asistencias de una fecha especifica.
    Compatible con frontend legacy.

    Args:
        fecha: Fecha en formato "dd-mm-yyyy" (ej: "09-01-2026")

    Returns:
        Lista de asistencias
    """
    if attendance_service is None:
        raise HTTPException(status_code=500, detail="Service no inicializado")

    try:
        print(f"[API] Consultando asistencias para: {fecha}")

        # Obtener asistencias
        asistencias = attendance_service.get_attendance_by_date(fecha)

        # Convertir a response model
        result = [UsersResponse(**asist) for asist in asistencias]

        print(f"[API] Encontradas {len(result)} asistencias")
        return result

    except Exception as e:
        print(f"[API] ❌ Error al consultar asistencias: {e}")
        raise HTTPException(status_code=500, detail=str(e))


@router.get("/asistencia/filtrada/{modelo_id}/{grupo_id}/{fecha}", response_model=List[UsersResponse])
async def get_attendance_filtered(modelo_id: int, grupo_id: str, fecha: str):
    """
    ✅ NUEVO: Obtiene asistencias filtradas por modelo, grupo y fecha.

    Args:
        modelo_id: ID del modelo
        grupo_id: ID del grupo
        fecha: Fecha en formato "dd-mm-yyyy"

    Returns:
        Lista de asistencias del grupo en esa fecha
    """
    if attendance_service is None or db_connector is None:
        raise HTTPException(status_code=500, detail="Service no inicializado")
```

Anexo 5. *Código relacionado con la captura de los datasets*



```
api > routes > datasets.py > DatasetSaveRequest
"""
API Routes: Gestión de Datasets
Captura y almacenamiento de datasets de imágenes faciales
"""

from fastapi import APIRouter, HTTPException
from pydantic import BaseModel, Field
from typing import List, Dict
import os
import base64
from pathlib import Path
from datetime import datetime


router = APIRouter(tags=["datasets"])


# ============================================
# MODELOS PYDANTIC
# ============================================

class DatasetSaveRequest(BaseModel):
    """Request para guardar dataset"""
    folder_name: str = Field(description="Nombre de la carpeta del dataset")
    images: List[str] = Field(description="Lista de imágenes en base64")
    user_data: Dict = Field(description="Datos del usuario")
```

```python
class DatasetSaveResponse(BaseModel):
    success: bool
    saved_count: int
    folder_path: str
    message: str


# ==========================================
# ENDPOINTS
# ==========================================

@router.post("/dataset/save", response_model=DatasetSaveResponse)
async def save_dataset(request: DatasetSaveRequest):
    """
    Guarda un dataset de imágenes en el sistema de archivos.

    Args:
        request: Datos del dataset (nombre carpeta, imágenes, datos usuario)

    Returns:
        Información sobre el guardado
    """
    try:
        # Crear directorio base de datasets si no existe
        datasets_dir = Path("datasets")
        datasets_dir.mkdir(exist_ok=True)

        # Crear subdirectorio para el usuario
```

api > routes > datasets.py > ...

```python
async def save_dataset(request: DatasetSaveRequest):
        # Crear subdirectorio para el usuario
        user_folder = datasets_dir / request.folder_name
        user_folder.mkdir(exist_ok=True)

        # Guardar cada imagen
        saved_count = 0

        for idx, img_base64 in enumerate(request.images, start=1):
            try:
                # Remover el prefijo  si existe
                if ',' in img_base64:
                    img_base64 = img_base64.split(',')[1]

                # Decodificar base64
                img_data = base64.b64decode(img_base64)

                # Nombre de archivo con timestamp para evitar colisiones
                filename = f"img_{idx:04d}.jpg"
                filepath = user_folder / filename

                # Guardar imagen
                with open(filepath, 'wb') as f:
                    f.write(img_data)

                saved_count += 1

            except Exception as e:
```

datasets.py 2

api > routes > datasets.py > save_dataset

```python
async def save_dataset(request: DatasetSaveRequest):
                print(f"[Dataset] ⚠️ Error al guardar imagen {idx}: {e}")
                continue

        # Guardar metadata
        metadata_file = user_folder / "metadata.txt"
        with open(metadata_file, 'w', encoding='utf-8') as f:
            f.write(f"Usuario: {request.user_data.get('fullname', 'Unknown')}\n")
            f.write(f"Código: {request.user_data.get('code', 'Unknown')}\n")
            f.write(f"User ID: {request.user_data.get('userId', 'Unknown')}\n")
            f.write(f"Fecha captura: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n")
            f.write(f"Total imágenes: {saved_count}\n")

        print(f"[Dataset] ✅ {saved_count} imágenes guardadas en {user_folder}")

        return DatasetSaveResponse(
            success=True,
            saved_count=saved_count,
            folder_path=str(user_folder),
            message=f"Dataset guardado exitosamente con {saved_count} imágenes"
        )

    except Exception as e:
        print(f"[Dataset] ❌ Error al guardar dataset: {e}")
        raise HTTPException(status_code=500, detail=str(e))


@router.get("/dataset/list")
```

pi > routes > datasets.py > save_dataset

```python

@router.get("/dataset/list")
async def list_datasets():
    """
    Lista todos los datasets disponibles.

    Returns:
        Lista de datasets con información básica
    """
    try:
        datasets_dir = Path("datasets")

        if not datasets_dir.exists():
            return []

        datasets = []

        for folder in datasets_dir.iterdir():
            if folder.is_dir():
                # Contar imágenes
                images = list(folder.glob("*.jpg")) + list(folder.glob("*.jpeg")) + list(folder.glob("*.png"))

                # Leer metadata si existe
                metadata_file = folder / "metadata.txt"
                metadata = {}
                if metadata_file.exists():
                    with open(metadata_file, 'r', encoding='utf-8') as f:
                        for line in f:
```


routes > datasets.py > list_datasets
async def list_datasets():
# Leer metadata si existe
metadata_file = folder / "metadata.txt"
metadata = {}
if metadata_file.exists():
with open(metadata_file, 'r', encoding='utf-8') as f:
for line in f:
if ':' in line:
key, value = line.strip().split(':', 1)
metadata[key.strip()] = value.strip()
datasets.append({
'folder_name': folder.name,
'folder_path': str(folder),
'image_count': len(images),
'metadata': metadata
})
return datasets
except Exception as e:
print(f"[Dataset] ❌ Error al listar datasets: {e}")
raise HTTPException(status_code=500, detail=str(e))
if __name__ == '__main__':
print("=== API ROUTES - DATASETS ===")

Anexo 6. *Código relacionado con el CRUD de grupos*

grupos.py 2

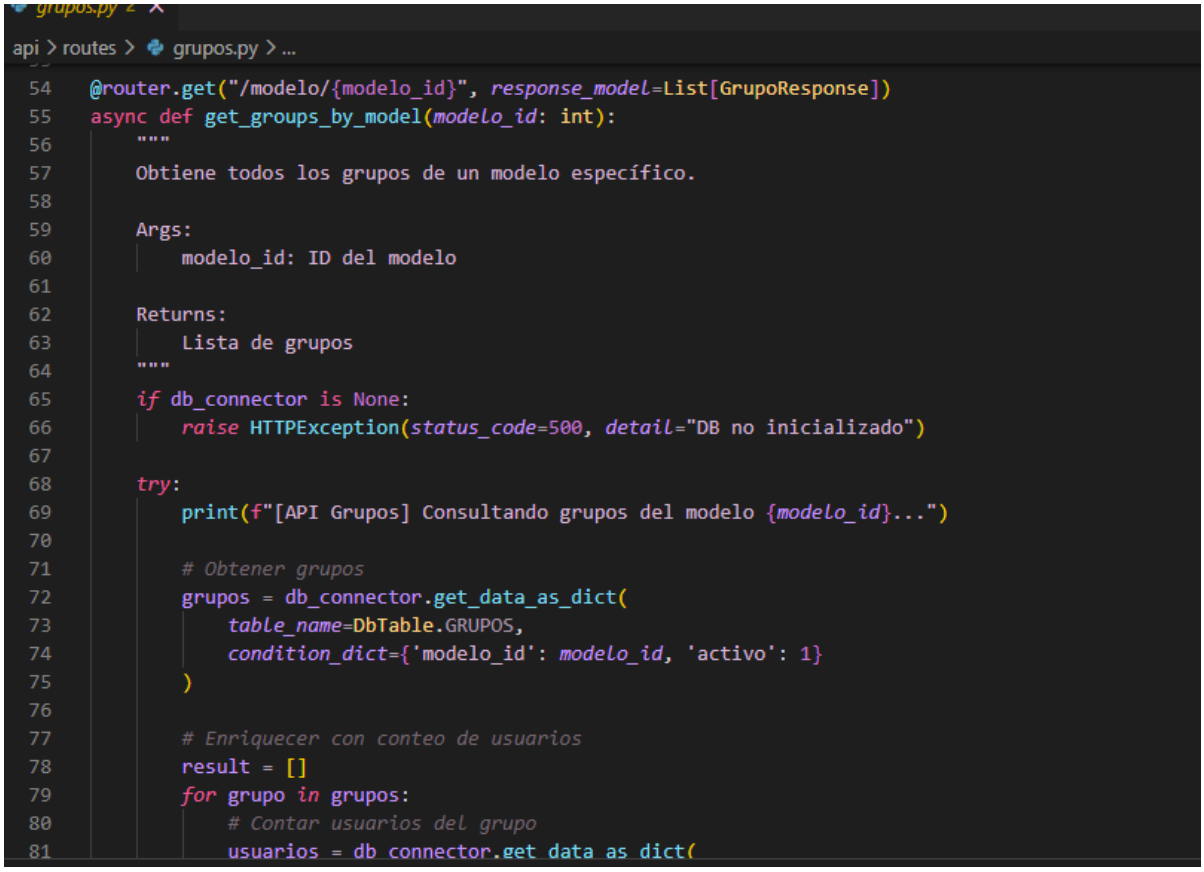
api > routes > grupos.py > ...

```python
"""
API Routes: Gestión de Grupos
CRUD completo para grupos de usuarios asociados a modelos
"""

from fastapi import APIRouter, HTTPException
from pydantic import BaseModel, Field
from typing import List, Optional
from datetime import datetime
import uuid

from database.db_models import DbTable


router = APIRouter(prefix="/grupos", tags=["grupos"])

# Instancia global (se inicializará en main.py)
db_connector = None


# ============================================
# MODELOS PYDANTIC
# ============================================

class GrupoCreate(BaseModel):
    """Request para crear un grupo"""
    modelo_id: int = Field(description="ID del modelo")
    nombre_grupo: str = Field(description="Nombre del grupo", min_length=1, max_length=100)
```

grupos.py 2

api > routes > grupos.py > ...

```python
class GrupoCreate(BaseModel):
    nombre_grupo: str = Field(description="Nombre del grupo", min_length=1, max_length=100)
    descripcion: Optional[str] = Field(description="Descripción opcional", default="")


class GrupoUpdate(BaseModel):
    """Request para actualizar un grupo"""
    nombre_grupo: Optional[str] = Field(description="Nuevo nombre", default=None)
    descripcion: Optional[str] = Field(description="Nueva descripción", default=None)
    activo: Optional[int] = Field(description="Estado (0=inactivo, 1=activo)", default=None)


class GrupoResponse(BaseModel):
    """Response de grupo"""
    idgrupo: str = Field(description="ID del grupo")
    modelo_id: int = Field(description="ID del modelo")
    nombre_grupo: str = Field(description="Nombre del grupo")
    descripcion: str = Field(description="Descripción")
    fecha_creacion: str = Field(description="Fecha de creación")
    activo: int = Field(description="Estado")
    usuarios_count: int = Field(description="Cantidad de usuarios", default=0)


# ============================================
# ENDPOINTS
# ============================================

@router.get("/modelo/{modelo_id}", response_model=List[GrupoResponse])
```



```
grupos.py 2
api > routes > grupos.py > ...
@router.get("/modelo/{modelo_id}", response_model=List[GrupoResponse])
async def get_groups_by_model(modelo_id: int):
    """
    Obtiene todos los grupos de un modelo específico.

    Args:
        modelo_id: ID del modelo

    Returns:
        Lista de grupos
    """
    if db_connector is None:
        raise HTTPException(status_code=500, detail="DB no inicializado")

    try:
        print(f"[API Grupos] Consultando grupos del modelo {modelo_id}...")

        # Obtener grupos
        grupos = db_connector.get_data_as_dict(
            table_name=DbTable.GRUPOS,
            condition_dict={'modelo_id': modelo_id, 'activo': 1}
        )

        # Enriquecer con conteo de usuarios
        result = []
        for grupo in grupos:
            # Contar usuarios del grupo
            usuarios = db_connector.get_data_as_dict(
```

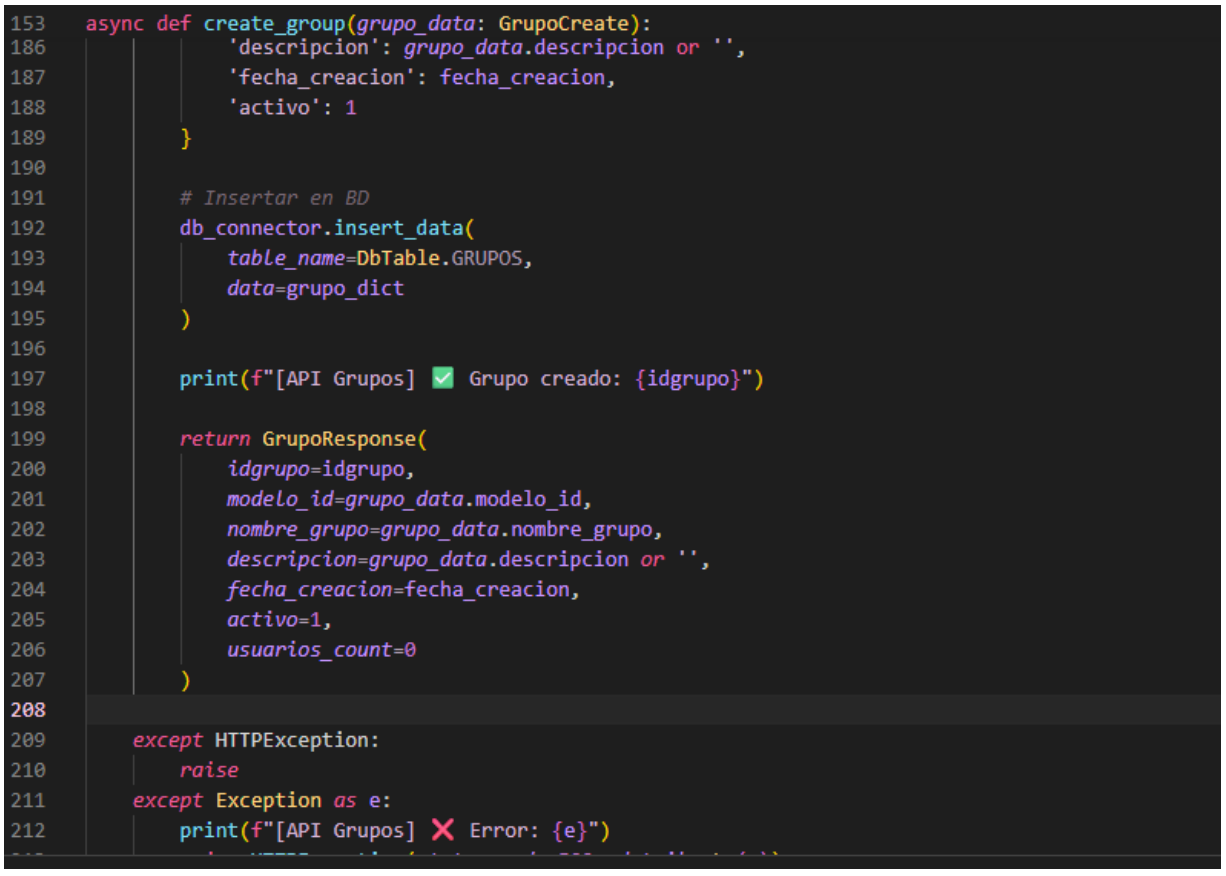
```
api > routes > grupos.py > get_groups_by_model
async def get_groups_by_model(modelo_id: int):
            usuarios = db_connector.get_data_as_dict(
                table_name=DbTable.USUARIOS,
                condition_dict={'grupo_id': grupo['idgrupo']}
            )

            result.append(GrupoResponse(
                idgrupo=grupo['idgrupo'],
                modelo_id=grupo['modelo_id'],
                nombre_grupo=grupo['nombre_grupo'],
                descripcion=grupo.get('descripcion', ''),
                fecha_creacion=grupo['fecha_creacion'],
                activo=grupo['activo'],
                usuarios_count=len(usuarios)
            ))

        print(f"[API Grupos] ✅ Encontrados {len(result)} grupos")
        return result

    except Exception as e:
        print(f"[API Grupos] ❌ Error: {e}")
        raise HTTPException(status_code=500, detail=str(e))


@router.get("/{idgrupo}", response_model=GrupoResponse)
async def get_group_by_id(idgrupo: str):
    """
    Obtiene un grupo específico por ID.
```

```python
async def get_group_by_id(idgrupo: str):
    Obtiene un grupo especifico por ID.

    Args:
        idgrupo: ID del grupo

    Returns:
        Información del grupo
    """
    if db_connector is None:
        raise HTTPException(status_code=500, detail="DB no inicializado")

    try:
        grupos = db_connector.get_data_as_dict(
            table_name=DbTable.GRUPOS,
            condition_dict={'idgrupo': idgrupo}
        )

        if not grupos:
            raise HTTPException(status_code=404, detail="Grupo no encontrado")

        grupo = grupos[0]

        # Contar usuarios
        usuarios = db_connector.get_data_as_dict(
            table_name=DbTable.USUARIOS,
            condition_dict={'grupo_id': idgrupo}
        )
```

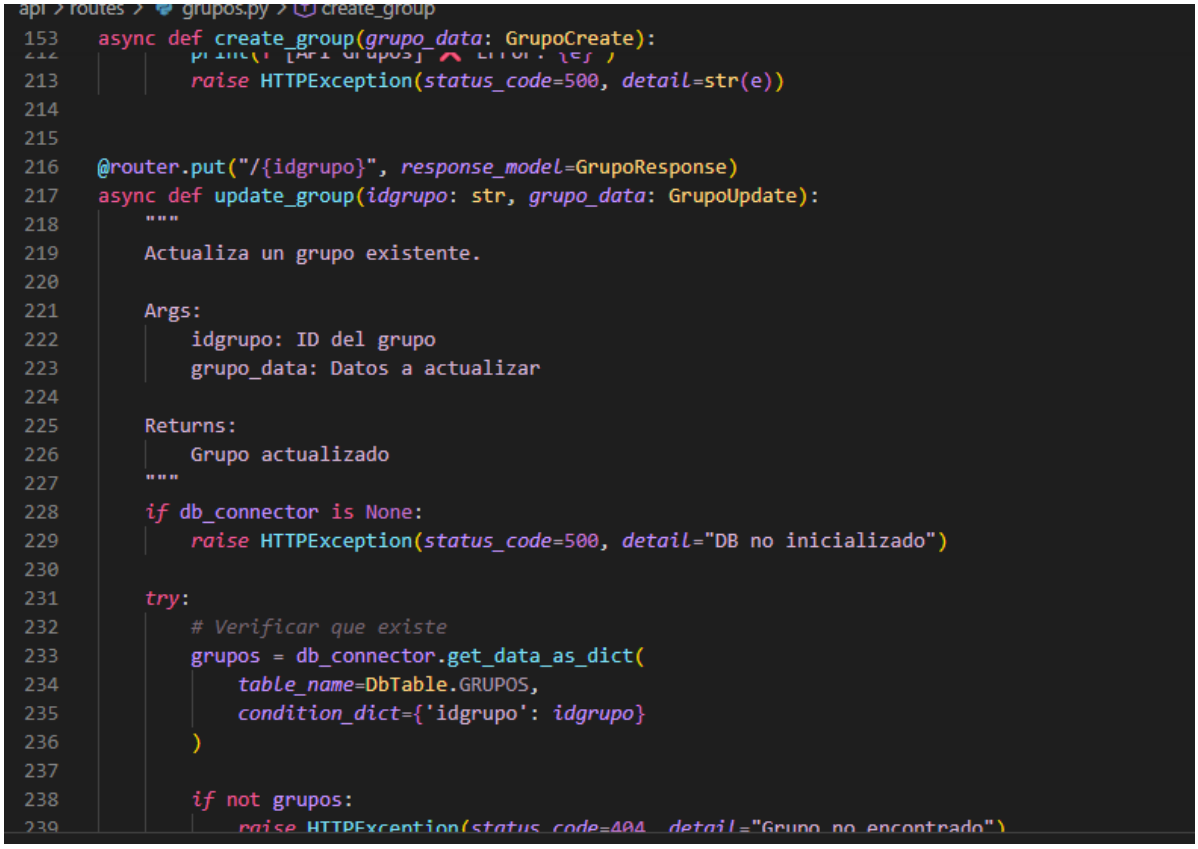
pi > routes > grupos.py > get_group_by_id

```python
async def get_group_by_id(idgrupo: str):
        '

        return GrupoResponse(
            idgrupo=grupo['idgrupo'],
            modelo_id=grupo['modelo_id'],
            nombre_grupo=grupo['nombre_grupo'],
            descripcion=grupo.get('descripcion', ''),
            fecha_creacion=grupo['fecha_creacion'],
            activo=grupo['activo'],
            usuarios_count=len(usuarios)
        )

    except HTTPException:
        raise
    except Exception as e:
        print(f"[API Grupos] ❌ Error: {e}")
        raise HTTPException(status_code=500, detail=str(e))


@router.post("/", response_model=GrupoResponse)
async def create_group(grupo_data: GrupoCreate):
    """
    Crea un nuevo grupo.

    Args:
        grupo_data: Datos del grupo a crear

    Returns:
```

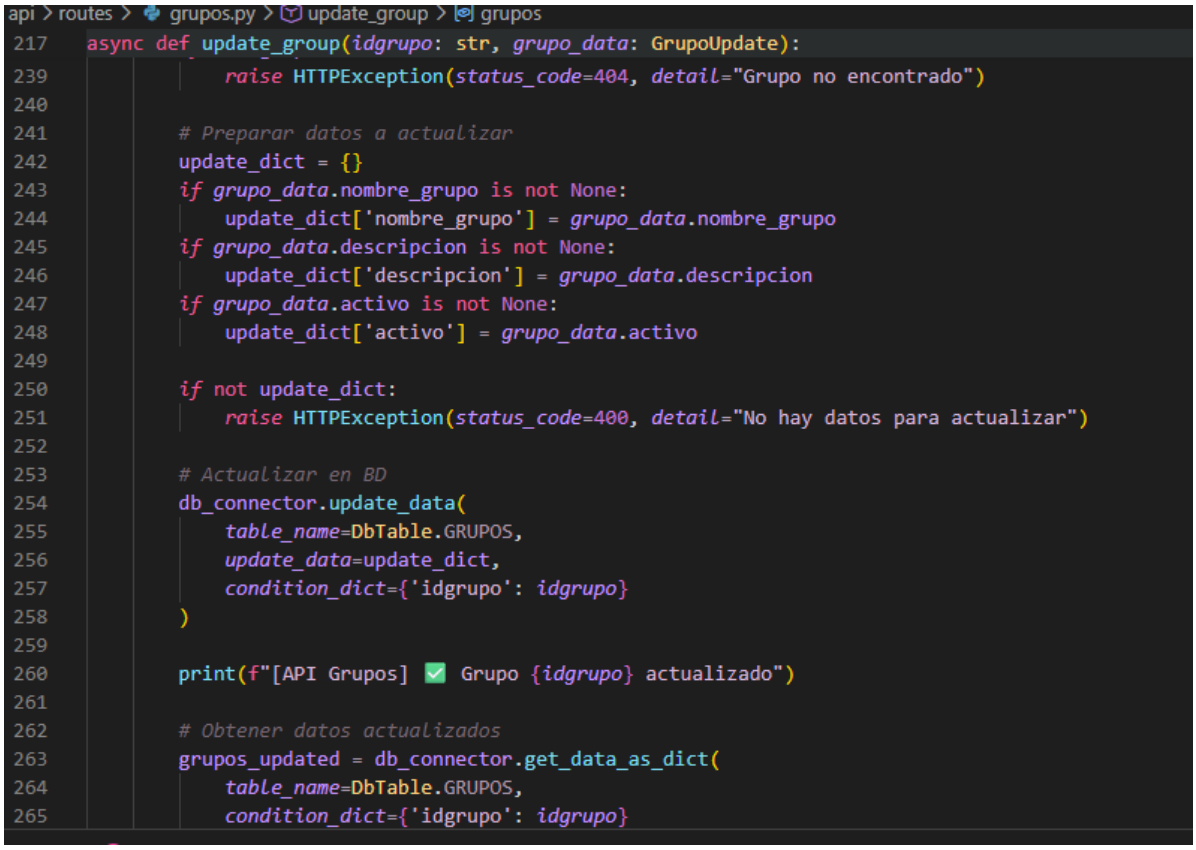
```
api > routes > grupos.py > create_group
153  async def create_group(grupo_data: GrupoCreate):
160      Returns:
161          Grupo creado
162      """
163      if db_connector is None:
164          raise HTTPException(status_code=500, detail="DB no inicializado")
165
166      try:
167          print(f"[API Grupos] Creando grupo: {grupo_data.nombre_grupo}")
168
169          # Verificar que el modelo existe
170          modelos = db_connector.get_data_as_dict(
171              table_name=DbTable.MODELOS,
172              condition_dict={'idmodelo': grupo_data.modelo_id}
173          )
174
175          if not modelos:
176              raise HTTPException(status_code=404, detail=f"Modelo {grupo_data.modelo_id} no existe")
177
178          # Generar ID y preparar datos
179          idgrupo = str(uuid.uuid4())
180          fecha_creacion = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
181
182          grupo_dict = {
183              'idgrupo': idgrupo,
184              'modelo_id': grupo_data.modelo_id,
185              'nombre_grupo': grupo_data.nombre_grupo,
186              'descripcion': grupo_data.descripcion or '',
```



```
153  async def create_group(grupo_data: GrupoCreate):
186              'descripcion': grupo_data.descripcion or '',
187              'fecha_creacion': fecha_creacion,
188              'activo': 1
189          }
190
191          # Insertar en BD
192          db_connector.insert_data(
193              table_name=DbTable.GRUPOS,
194              data=grupo_dict
195          )
196
197          print(f"[API Grupos] ✅ Grupo creado: {idgrupo}")
198
199          return GrupoResponse(
200              idgrupo=idgrupo,
201              modelo_id=grupo_data.modelo_id,
202              nombre_grupo=grupo_data.nombre_grupo,
203              descripcion=grupo_data.descripcion or '',
204              fecha_creacion=fecha_creacion,
205              activo=1,
206              usuarios_count=0
207          )
208
209      except HTTPException:
210          raise
211      except Exception as e:
212          print(f"[API Grupos] ❌ Error: {e}")
```



```
api > routes > grupos.py > create_group
153  async def create_group(grupo_data: GrupoCreate):
212          print(f"[API Grupos] ❌ Error: {e}")
213          raise HTTPException(status_code=500, detail=str(e))
214
215
216  @router.put("/{idgrupo}", response_model=GrupoResponse)
217  async def update_group(idgrupo: str, grupo_data: GrupoUpdate):
218      """
219      Actualiza un grupo existente.
220
221      Args:
222          idgrupo: ID del grupo
223          grupo_data: Datos a actualizar
224
225      Returns:
226          Grupo actualizado
227      """
228      if db_connector is None:
229          raise HTTPException(status_code=500, detail="DB no inicializado")
230
231      try:
232          # Verificar que existe
233          grupos = db_connector.get_data_as_dict(
234              table_name=DbTable.GRUPOS,
235              condition_dict={'idgrupo': idgrupo}
236          )
237
238          if not grupos:
239              raise HTTPException(status_code=404, detail="Grupo no encontrado")
```



```
api > routes > grupos.py > update_group > grupos
217  async def update_group(idgrupo: str, grupo_data: GrupoUpdate):
239              raise HTTPException(status_code=404, detail="Grupo no encontrado")
240
241          # Preparar datos a actualizar
242          update_dict = {}
243          if grupo_data.nombre_grupo is not None:
244              update_dict['nombre_grupo'] = grupo_data.nombre_grupo
245          if grupo_data.descripcion is not None:
246              update_dict['descripcion'] = grupo_data.descripcion
247          if grupo_data.activo is not None:
248              update_dict['activo'] = grupo_data.activo
249
250          if not update_dict:
251              raise HTTPException(status_code=400, detail="No hay datos para actualizar")
252
253          # Actualizar en BD
254          db_connector.update_data(
255              table_name=DbTable.GRUPOS,
256              update_data=update_dict,
257              condition_dict={'idgrupo': idgrupo}
258          )
259
260          print(f"[API Grupos] ✅ Grupo {idgrupo} actualizado")
261
262          # Obtener datos actualizados
263          grupos_updated = db_connector.get_data_as_dict(
264              table_name=DbTable.GRUPOS,
265              condition_dict={'idgrupo': idgrupo}
```

api > routes > grupos.py > update_group

```python
    async def update_group(idgrupo: str, grupo_data: GrupoUpdate):
                condition_dict={'idgrupo': idgrupo}
            )

            grupo = grupos_updated[0]

            # Contar usuarios
            usuarios = db_connector.get_data_as_dict(
                table_name=DbTable.USUARIOS,
                condition_dict={'grupo_id': idgrupo}
            )

            return GrupoResponse(
                idgrupo=grupo['idgrupo'],
                modelo_id=grupo['modelo_id'],
                nombre_grupo=grupo['nombre_grupo'],
                descripcion=grupo.get('descripcion', ''),
                fecha_creacion=grupo['fecha_creacion'],
                activo=grupo['activo'],
                usuarios_count=len(usuarios)
            )

        except HTTPException:
            raise
        except Exception as e:
            print(f"[API Grupos] ❌ Error: {e}")
            raise HTTPException(status_code=500, detail=str(e))

```

api > routes > grupos.py > update_group

```python
@router.delete("/{idgrupo}")
async def delete_group(idgrupo: str):
    """
    Desactiva un grupo (soft delete).
    No elimina los usuarios, solo desactiva el grupo.

    Args:
        idgrupo: ID del grupo

    Returns:
        Confirmación
    """
    if db_connector is None:
        raise HTTPException(status_code=500, detail="DB no inicializado")

    try:
        # Verificar que existe
        grupos = db_connector.get_data_as_dict(
            table_name=DbTable.GRUPOS,
            condition_dict={'idgrupo': idgrupo}
        )

        if not grupos:
            raise HTTPException(status_code=404, detail="Grupo no encontrado")

        # Soft delete
        db_connector.update_data(
            table_name=DbTable.GRUPOS,
```

api > routes > grupos.py > update_group

```python
294    async def delete_group(idgrupo: str):
320                table_name=DbTable.GRUPOS,
321                update_data={'activo': 0},
322                condition_dict={'idgrupo': idgrupo}
323            )
324
325            print(f"[API Grupos] ✅ Grupo {idgrupo} desactivado")
326
327            return {
328                "success": True,
329                "message": "Grupo desactivado correctamente",
330                "idgrupo": idgrupo
331            }
332
333        except HTTPException:
334            raise
335        except Exception as e:
336            print(f"[API Grupos] ❌ Error: {e}")
337            raise HTTPException(status_code=500, detail=str(e))
338
339
340    @router.get("/")
341    async def get_all_groups(activo: Optional[int] = 1):
342        """
343        Obtiene todos los grupos del sistema.
344
345        Args:
346            activo: Filtrar por estado (1=activos, 0=inactivos, None=todos)
347
```

api > routes > grupos.py > get_all_groups

```python
341    async def get_all_groups(activo: Optional[int] = 1):
347
348        Returns:
349            Lista de grupos
350        """
351        if db_connector is None:
352            raise HTTPException(status_code=500, detail="DB no inicializado")
353
354        try:
355            condition = {}
356            if activo is not None:
357                condition['activo'] = activo
358
359            grupos = db_connector.get_data_as_dict(
360                table_name=DbTable.GRUPOS,
361                condition_dict=condition if condition else None
362            )
363
364            result = []
365            for grupo in grupos:
366                usuarios = db_connector.get_data_as_dict(
367                    table_name=DbTable.USUARIOS,
368                    condition_dict={'grupo_id': grupo['idgrupo']}
369                )
370
371                result.append(GrupoResponse(
372                    idgrupo=grupo['idgrupo'],
373                    modelo_id=grupo['modelo_id'],
```

```python
async def get_all_groups(activo: Optional[int] = 1):
                    modelo_id=grupo['modelo_id'],
                    nombre_grupo=grupo['nombre_grupo'],
                    descripcion=grupo.get('descripcion', ''),
                    fecha_creacion=grupo['fecha_creacion'],
                    activo=grupo['activo'],
                    usuarios_count=len(usuarios)
                ))

            return result

        except Exception as e:
            print(f"[API Grupos] ❌ Error: {e}")
            raise HTTPException(status_code=500, detail=str(e))


if __name__ == '__main__':
    print("=== API ROUTES - GRUPOS ===")
    print("\nEndpoints disponibles:")
    print("  GET    /grupos/modelo/{modelo_id} - Grupos de un modelo")
    print("  GET    /grupos/{idgrupo}          - Grupo específico")
    print("  POST   /grupos/                   - Crear grupo")
    print("  PUT    /grupos/{idgrupo}          - Actualizar grupo")
    print("  DELETE /grupos/{idgrupo}          - Desactivar grupo")
    print("  GET    /grupos/                   - Todos los grupos")
```

Anexo 7. *Código relacionado con el cambio dinámico de modelos*

```python
"""
API Route: Cambio de Modelo
Permite cambiar dinámicamente entre diferentes modelos de reconocimiento
"""

from fastapi import APIRouter, HTTPException
from pydantic import BaseModel
from typing import List, Optional

from core.model_manager import model_manager


router = APIRouter(prefix="/model", tags=["model"])


# ============================================
# MODELOS PYDANTIC
# ============================================

class ModelInfo(BaseModel):
    """Información de un modelo"""
    model_id: int
    model_name: str
    embedding_dim: int
    threshold: float
    is_current: bool
    is_loaded: bool


class SwitchModelRequest(BaseModel):
    """Request para cambiar de modelo"""
    model_id: int


class SwitchModelResponse(BaseModel):
    """Response del cambio de modelo"""
    success: bool
    message: str
    model_info: Optional[dict] = None


# ============================================
# ENDPOINTS
# ============================================

@router.get("/available", response_model=List[ModelInfo])
async def get_available_models():
    """
```

```python
@router.get("/available", response_model=List[ModelInfo])
async def get_available_models():
    """
    Obtiene la lista de modelos disponibles.

    Returns:
        Lista de modelos con su información
    """
    try:
        models = model_manager.get_available_models()
        return [ModelInfo(**model) for model in models]
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Error al obtener modelos: {str(e)}")


@router.get("/current")
async def get_current_model():
    """
    Obtiene información del modelo actualmente activo.

    Returns:
        Información del modelo actual o None
    """
    try:
        model_info = model_manager.get_current_model_info()

        if model_info is None:
            return {
                "success": False,
                "message": "No hay ningún modelo cargado",
                "model_info": None
            }

        return {
            "success": True,
            "message": "Modelo actual obtenido",
            "model_info": model_info
        }
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Error al obtener modelo actual: {str(e)}")


@router.post("/switch", response_model=SwitchModelResponse)
async def switch_model(request: SwitchModelRequest):
    """
    Cambia al modelo especificado.

    Args:
```

api > routes > model_switch.py > get_current_model

```python
async def switch_model(request: SwitchModelRequest):
        request: Contiene el ID del modelo a activar (1=EdgeFace-s, 2=MobileFaceNet, 3=EdgeFace-xs)

    Returns:
        Confirmación del cambio y info del nuevo modelo
    """
    try:
        model_id = request.model_id

        # Validar ID
        if model_id not in model_manager.available_models:
            available_ids = list(model_manager.available_models.keys())
            raise HTTPException(
                status_code=400,
                detail=f"Modelo {model_id} no existe. Disponibles: {available_ids}"
            )

        # Cambiar modelo
        success = model_manager.switch_model(model_id)

        if success:
            model_info = model_manager.get_current_model_info()
            return SwitchModelResponse(
                success=True,
                message=f"Modelo {model_id} activado exitosamente",
                model_info=model_info
            )
        else:
```

api > routes > model_switch.py > switch_model

```python
async def switch_model(request: SwitchModelRequest):
            else:
                return SwitchModelResponse(
                    success=False,
                    message=f"Error al cambiar al modelo {model_id}",
                    model_info=None
                )

    except HTTPException:
        raise
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Error al cambiar modelo: {str(e)}")


@router.post("/reload")
async def reload_current_model():
    """
    Recarga el modelo actual (útil si hubo un error).

    Returns:
        Confirmación de la recarga
    """
    try:
        current_model_id = model_manager.current_model_id

        if current_model_id is None:
            return {
                "success": False,
                "message": "No hay modelo activo para recargar"
```

api > routes > model_switch.py > reload_current_model

```python
async def reload_current_model():
                    "message": "No hay modelo activo para recargar"
                }

            # Recargar (equivalente a switch al mismo modelo)
            success = model_manager.switch_model(current_model_id)

            if success:
                return {
                    "success": True,
                    "message": f"Modelo {current_model_id} recargado exitosamente"
                }
            else:
                return {
                    "success": False,
                    "message": f"Error al recargar modelo {current_model_id}"
                }

        except Exception as e:
            raise HTTPException(status_code=500, detail=f"Error al recargar modelo: {str(e)}")


@router.get("/info/{model_id}")
async def get_model_info(model_id: int):
    """
    Obtiene información detallada de un modelo específico.

    Args:
```

api > routes > model_switch.py > get_model_info

```python
async def get_model_info(model_id: int):
        model_id: ID del modelo

    Returns:
        Información del modelo
    """
    try:
        if model_id not in model_manager.available_models:
            raise HTTPException(
                status_code=404,
                detail=f"Modelo {model_id} no encontrado"
            )

        # Crear instancia temporal para obtener info
        model_class = model_manager.available_models[model_id]
        temp_instance = model_class(model_id=model_id)

        return {
            "model_id": model_id,
            "model_name": temp_instance.model_name,
            "embedding_dimension": temp_instance.get_embedding_dimension(),
            "recommended_threshold": temp_instance.get_recommended_threshold(),
            "is_current": (model_id == model_manager.current_model_id),
            "is_loaded": (
                model_id == model_manager.current_model_id and
                model_manager.current_model is not None and
                model_manager.current_model.is_loaded
            )
        }

    except HTTPException:
        raise
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Error al obtener info del modelo: {str(e)}")

```

```python
    except HTTPException:
        raise
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Error al obtener info del modelo: {str(e)}")


# Test
if __name__ == '__main__':
    print("=== API ROUTES - MODEL SWITCH ===")
    print("\nEndpoints disponibles:")
    print("  GET  /model/available        - Lista de modelos")
    print("  GET  /model/current          - Modelo actual")
    print("  POST /model/switch           - Cambiar modelo")
    print("  POST /model/reload           - Recargar modelo")
    print("  GET  /model/info/{model_id}  - Info de modelo")

```

Anexo 8. *Código relacionado con la Gestión de usuarios*

pi > routes > users.py > ImageResponse

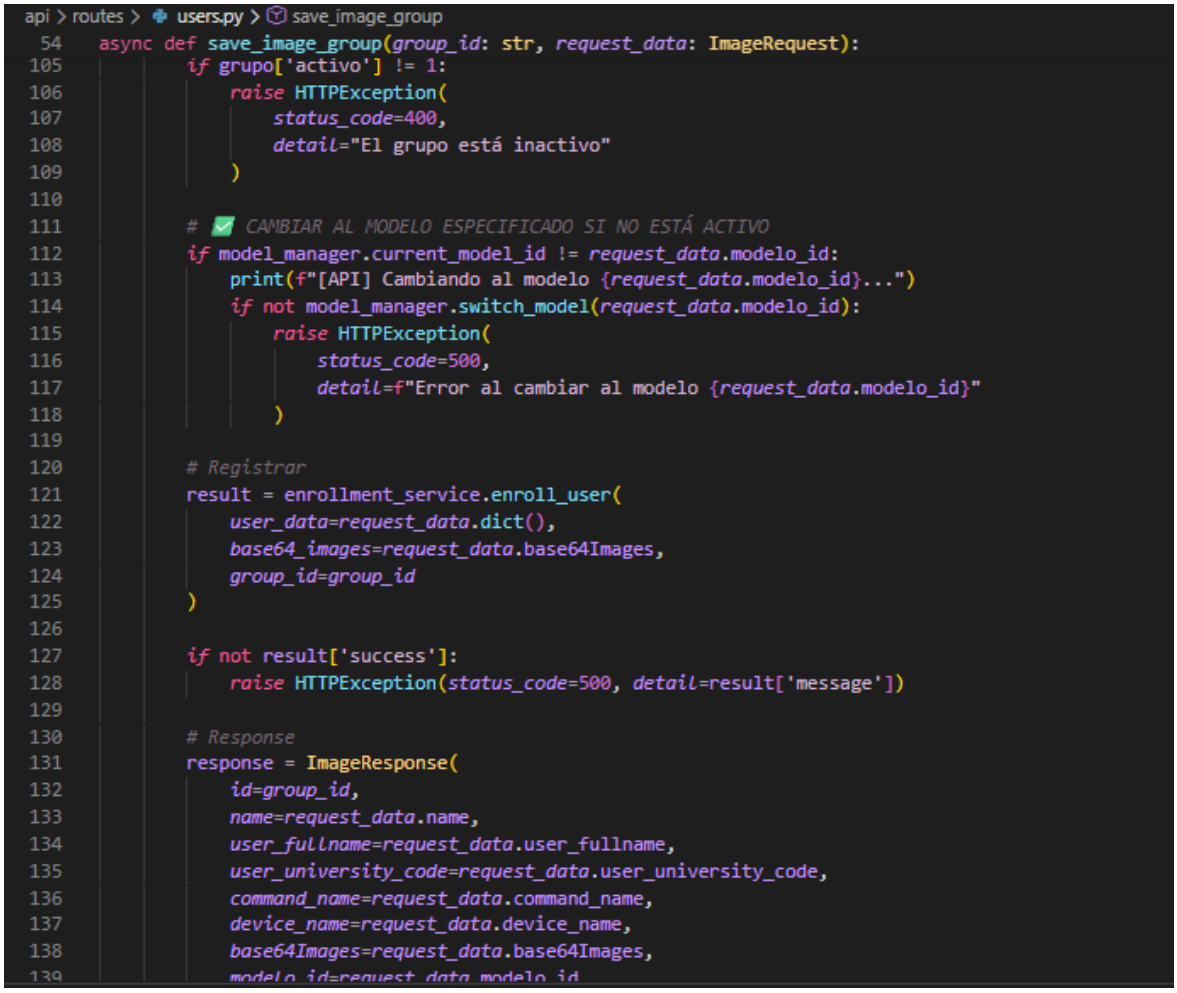
```python
"""
API Routes: Gestión de Usuarios CON GRUPOS
Registro de usuarios asociados a grupos de modelos
"""

from fastapi import APIRouter, HTTPException, Response
from pydantic import BaseModel, Field
from typing import List, Optional

from core.enrollment_service import EnrollmentService
from core.model_manager import model_manager


router = APIRouter(tags=["users"])

# Instancia global (se inicializará en main.py)
enrollment_service: Optional[EnrollmentService] = None


# ============================================
# MODELOS PYDANTIC
# ============================================

class ImageRequest(BaseModel):
    """Request para registrar usuario"""
    name: str = Field(description="Username")
    user_fullname: str = Field(description="Nombre completo")
    user_university_code: str = Field(description="Código universitario")
    command_name: str = Field(description="Comando", default="ON")
    device_name: str = Field(description="Dispositivo", default="GREEN_LED")
    base64Images: List[str] = Field(description="Lista de imágenes base64")
    modelo_id: int = Field(description="ID del modelo a utilizar")
    grupo_id: str = Field(description="ID del grupo al que pertenece")  # NUEVO


class ImageResponse(BaseModel):
```

api > routes > users.py > ImageRequest

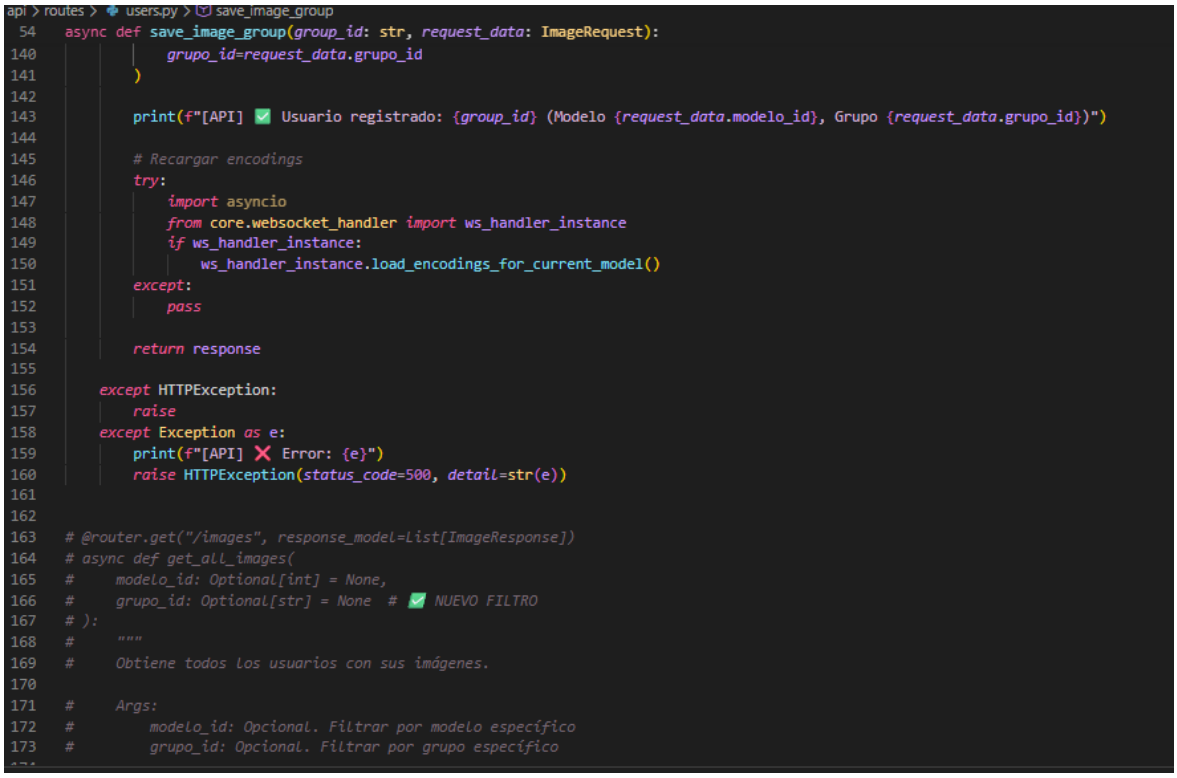
```python
class ImageResponse(BaseModel):
    """Response de usuario"""
    id: str
    name: str
    user_fullname: str
    user_university_code: str
    command_name: str = ""
    device_name: str = ""
    base64Images: List[str] = []
    modelo_id: int = 1
    grupo_id: str = ""  # ✅ NUEVO


# ============================================
# ENDPOINTS
# ============================================

@router.post("/save/image_group/{group_id}", response_model=ImageResponse)
async def save_image_group(group_id: str, request_data: ImageRequest):
    """
    Registra un nuevo usuario con sus imágenes.
    Compatible con frontend legacy.

    Args:
        group_id: UUID del usuario
        request_data: Datos e imágenes del usuario (incluye modelo_id y grupo_id)

    Returns:
        Usuario registrado
    """
    if enrollment_service is None:
        raise HTTPException(status_code=500, detail="Service no inicializado")

    try:
        print(f"[API] Registrando usuario: {request_data.user_fullname}")
```

api > routes > users.py > save_image_group

```python
async def save_image_group(group_id: str, request_data: ImageRequest):
        print(f"[API] Modelo: {request_data.modelo_id}, Grupo: {request_data.grupo_id}")

        if len(request_data.base64Images) == 0:
            raise HTTPException(status_code=400, detail="Debe proporcionar imágenes")

        # ✅ VALIDAR QUE EL MODELO EXISTE
        if request_data.modelo_id not in model_manager.available_models:
            available_ids = list(model_manager.available_models.keys())
            raise HTTPException(
                status_code=400,
                detail=f"Modelo {request_data.modelo_id} no existe. Disponibles: {available_ids}"
            )

        # ✅ VALIDAR QUE EL GRUPO EXISTE Y PERTENECE AL MODELO
        from database.db_models import DbTable

        grupos = enrollment_service.db_connector.get_data_as_dict(
            table_name=DbTable.GRUPOS,
            condition_dict={'idgrupo': request_data.grupo_id}
        )

        if not grupos:
            raise HTTPException(
                status_code=404,
                detail=f"Grupo {request_data.grupo_id} no existe"
            )

        grupo = grupos[0]
        if grupo['modelo_id'] != request_data.modelo_id:
            raise HTTPException(
                status_code=400,
                detail=f"El grupo pertenece al modelo {grupo['modelo_id']}, no al modelo {request_data.modelo_id}"
            )
```



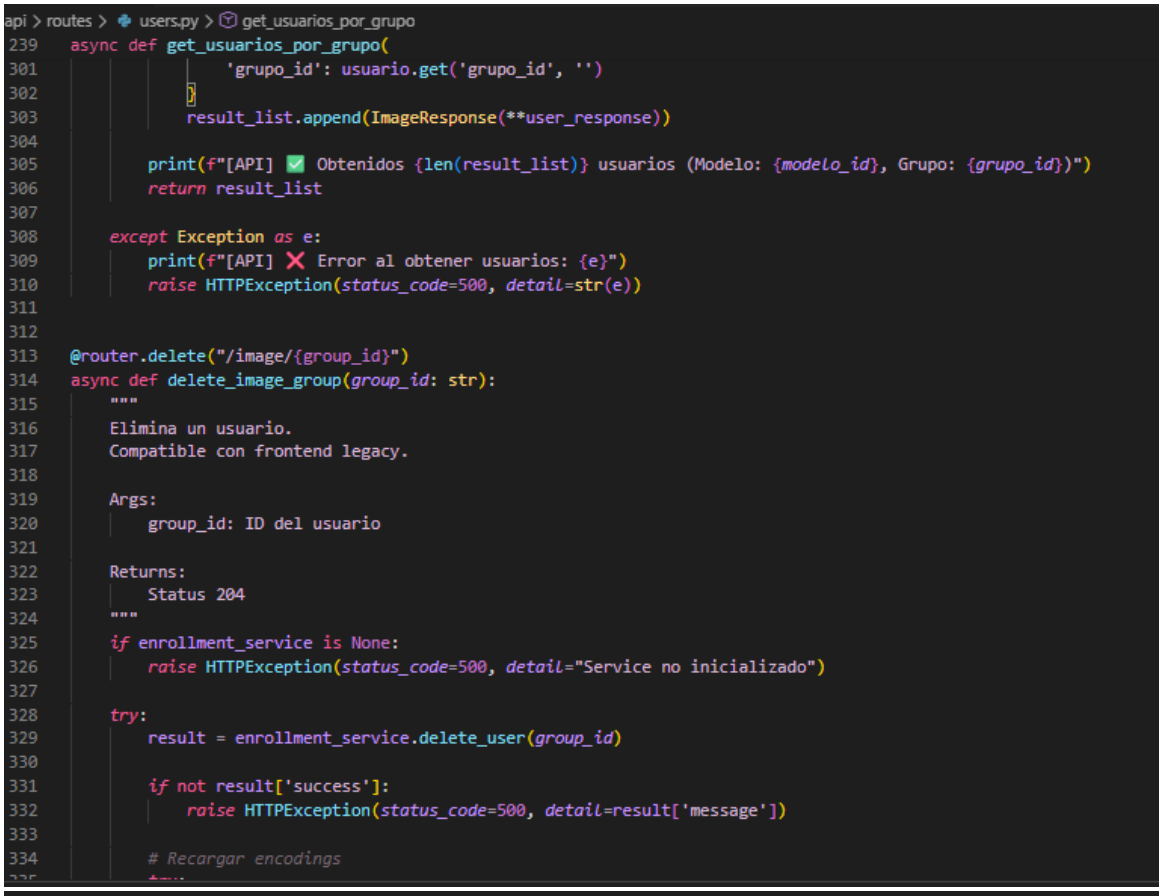
```
api > routes > users.py > save_image_group
 54   async def save_image_group(group_id: str, request_data: ImageRequest):
105          if grupo['activo'] != 1:
106              raise HTTPException(
107                  status_code=400,
108                  detail="El grupo está inactivo"
109              )
110
111          # ✅ CAMBIAR AL MODELO ESPECIFICADO SI NO ESTÁ ACTIVO
112          if model_manager.current_model_id != request_data.modelo_id:
113              print(f"[API] Cambiando al modelo {request_data.modelo_id}...")
114              if not model_manager.switch_model(request_data.modelo_id):
115                  raise HTTPException(
116                      status_code=500,
117                      detail=f"Error al cambiar al modelo {request_data.modelo_id}"
118                  )
119
120          # Registrar
121          result = enrollment_service.enroll_user(
122              user_data=request_data.dict(),
123              base64_images=request_data.base64Images,
124              group_id=group_id
125          )
126
127          if not result['success']:
128              raise HTTPException(status_code=500, detail=result['message'])
129
130          # Response
131          response = ImageResponse(
132              id=group_id,
133              name=request_data.name,
134              user_fullname=request_data.user_fullname,
135              user_university_code=request_data.user_university_code,
136              command_name=request_data.command_name,
137              device_name=request_data.device_name,
138              base64Images=request_data.base64Images,
139              modelo_id=request_data.modelo_id
```



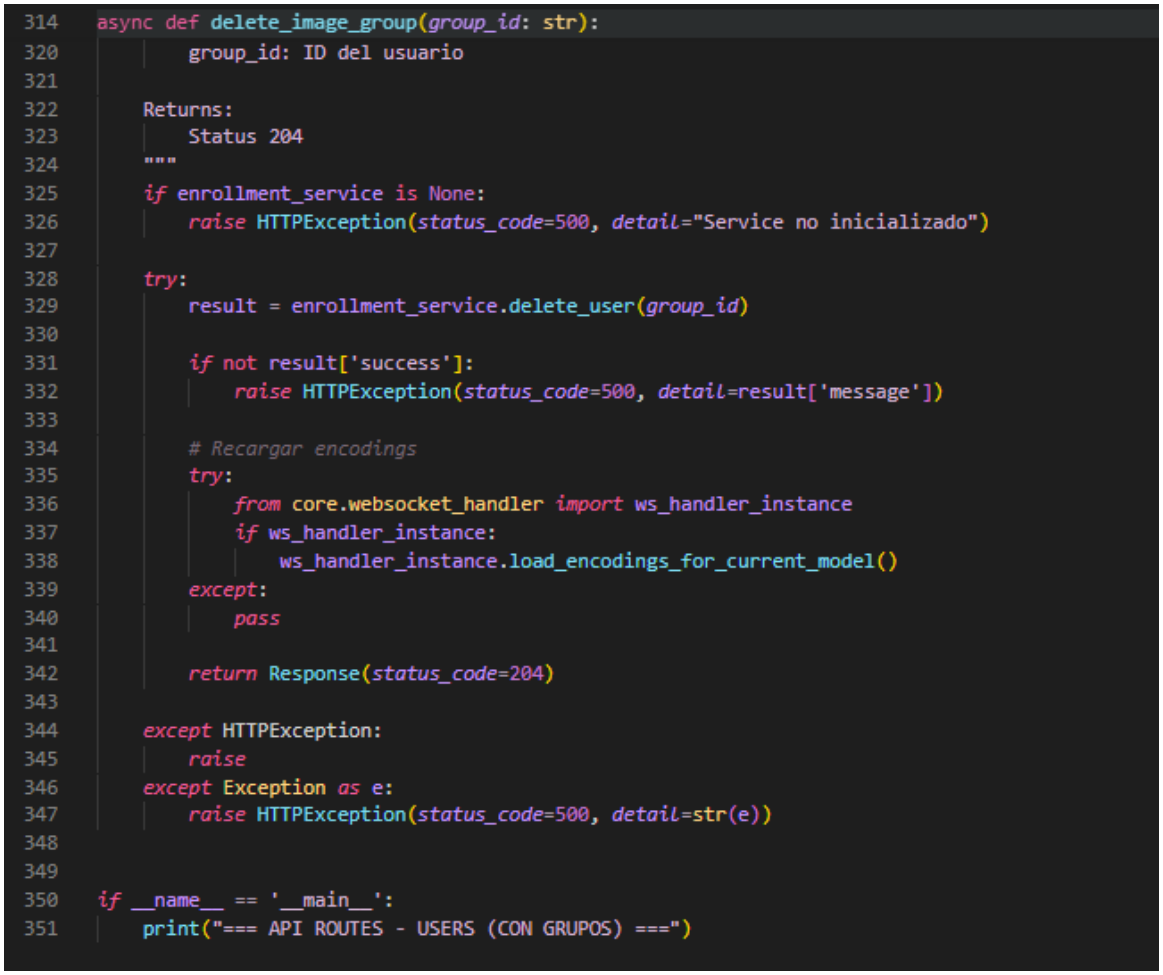
```
api > routes > users.py > save_image_group
 54   async def save_image_group(group_id: str, request_data: ImageRequest):
140              grupo_id=request_data.grupo_id
141          )
142
143          print(f"[API] ✅ Usuario registrado: {group_id} (Modelo {request_data.modelo_id}, Grupo {request_data.grupo_id})")
144
145          # Recargar encodings
146          try:
147              import asyncio
148              from core.websocket_handler import ws_handler_instance
149              if ws_handler_instance:
150                  ws_handler_instance.load_encodings_for_current_model()
151          except:
152              pass
153
154          return response
155
156      except HTTPException:
157          raise
158      except Exception as e:
159          print(f"[API] ❌ Error: {e}")
160          raise HTTPException(status_code=500, detail=str(e))
161
162
163  # @router.get("/images", response_model=List[ImageResponse])
164  # async def get_all_images(
165  #     modelo_id: Optional[int] = None,
166  #     grupo_id: Optional[str] = None  # ✅ NUEVO FILTRO
167  # ):
168  #     """
169  #     Obtiene todos los usuarios con sus imágenes.
170
171  #     Args:
172  #         modelo_id: Opcional. Filtrar por modelo específico
173  #         grupo_id: Opcional. Filtrar por grupo específico
```

```
routes > users.py > ...
# ========================================
# NUEVO ENDPOINT: OBTENER USUARIOS POR MODELO Y GRUPO
# ========================================

@router.get("/usuarios", response_model=List[ImageResponse])
async def get_usuarios_por_grupo(
    modelo_id: Optional[int] = None,
    grupo_id: Optional[str] = None
):
    """
    ✅ Obtiene usuarios registrados, con filtros opcionales por modelo y grupo.

    Args:
        modelo_id: Opcional. ID del modelo para filtrar
        grupo_id: Opcional. ID del grupo para filtrar

    Returns:
        Lista de usuarios con sus datos (incluyendo imágenes base64)
    """
    if enrollment_service is None:
        raise HTTPException(status_code=500, detail="Service no inicializado")

    try:
        from database.db_models import DbTable

        # ✅ CONSTRUIR FILTRO DINÁMICO
        condition = {}
        if modelo_id is not None:
            condition['modelo_id'] = modelo_id
        if grupo_id is not None:
            condition['grupo_id'] = grupo_id

        # Obtener usuarios con los filtros
        usuarios = enrollment_service.db_connector.get_data_as_dict(
            table_name=DbTable.USUARIOS,
```

```
api > routes > users.py > ...
async def get_usuarios_por_grupo(
        # Obtener usuarios con los filtros
        usuarios = enrollment_service.db_connector.get_data_as_dict(
            table_name=DbTable.USUARIOS,
            condition_dict=condition if condition else None
        )

        if not usuarios:
            print(f"[API] No hay usuarios para los filtros: modelo={modelo_id}, grupo={grupo_id}")
            return []

        # Obtener imágenes
        imagenes = enrollment_service.db_connector.get_data_as_dict(
            table_name=DbTable.BASE64_IMAGES
        )

        # Mapear usuarios con sus imágenes
        # La relación es: usuario.user_id <-> imagen.group_id
        result_list = []

        for usuario in usuarios:
            user_id = usuario.get('user_id')

            # Obtener imágenes del usuario
            user_images = [img['data'] for img in imagenes if img.get('group_id') == user_id]

            # Crear response
            user_response = {
                'id': user_id,
                'name': usuario.get('username', ''),
                'user_fullname': usuario.get('nombre_completo', ''),
                'user_university_code': usuario.get('codigo_universitario', ''),
                'command_name': 'ON',
                'device_name': 'GREEN_LED',
                'base64Images': user_images,
                'modelo_id': usuario.get('modelo_id', 1),
```



```
api > routes > users.py > get_usuarios_por_grupo
async def get_usuarios_por_grupo(
                'grupo_id': usuario.get('grupo_id', '')
            }
            result_list.append(ImageResponse(**user_response))

        print(f"[API] ✅ Obtenidos {len(result_list)} usuarios (Modelo: {modelo_id}, Grupo: {grupo_id})")
        return result_list

    except Exception as e:
        print(f"[API] ❌ Error al obtener usuarios: {e}")
        raise HTTPException(status_code=500, detail=str(e))


@router.delete("/image/{group_id}")
async def delete_image_group(group_id: str):
    """
    Elimina un usuario.
    Compatible con frontend legacy.

    Args:
        group_id: ID del usuario

    Returns:
        Status 204
    """
    if enrollment_service is None:
        raise HTTPException(status_code=500, detail="Service no inicializado")

    try:
        result = enrollment_service.delete_user(group_id)

        if not result['success']:
            raise HTTPException(status_code=500, detail=result['message'])

        # Recargar encodings
```



```
async def delete_image_group(group_id: str):
        group_id: ID del usuario

    Returns:
        Status 204
    """
    if enrollment_service is None:
        raise HTTPException(status_code=500, detail="Service no inicializado")

    try:
        result = enrollment_service.delete_user(group_id)

        if not result['success']:
            raise HTTPException(status_code=500, detail=result['message'])

        # Recargar encodings
        try:
            from core.websocket_handler import ws_handler_instance
            if ws_handler_instance:
                ws_handler_instance.load_encodings_for_current_model()
        except:
            pass

        return Response(status_code=204)

    except HTTPException:
        raise
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))


if __name__ == '__main__':
    print("=== API ROUTES - USERS (CON GRUPOS) ===")
```

Anexo 9. *Código relacionado con el Servicio de asistencias*

ore > attendance_service.py > ...

```python
"""
Attendance Service - Gestión de Asistencias
Guarda, consulta y valida asistencias
"""

import uuid
from datetime import datetime, timezone, timedelta
from typing import List, Dict, Optional

from database.db_models import DbTable
from core.model_manager import model_manager


class AttendanceService:
    """
    Servicio para gestionar asistencias de usuarios.
    """

    def __init__(self, db_connector):
        """
        Args:
            db_connector: Instancia de SQLiteConnector
        """
        self.db_connector = db_connector

    def validate_attendance_state(self, fecha_actual: datetime) -> str:
        """
        Determina el estado de asistencia según la hora.

        Args:
            fecha_actual: Datetime actual

        Returns:
            Estado: "ASISTIÓ", "TARDANZA", "FALTA", "FIN DE SEMANA"
        """
```

core > attendance_service.py > ...

```python
class AttendanceService:
    def validate_attendance_state(self, fecha_actual: datetime) -> str:
        """
        # Verificar si es fin de semana
        if fecha_actual.weekday() in [5, 6]:  # Sábado=5, Domingo=6
            return "FIN DE SEMANA"

        # Horas de referencia (ajusta según tus necesidades)
        hora_asistio = datetime(
            fecha_actual.year, fecha_actual.month, fecha_actual.day,
            8, 0, 0  # 8:00 AM
        )
        hora_tardanza_inicio = datetime(
            fecha_actual.year, fecha_actual.month, fecha_actual.day,
            8, 12, 0  # 8:12 AM
        )
        hora_tardanza_fin = datetime(
            fecha_actual.year, fecha_actual.month, fecha_actual.day,
            8, 15, 0  # 8:15 AM
        )

        # Determinar estado
        if fecha_actual < hora_tardanza_inicio:
            return "ASISTIÓ"
        elif hora_tardanza_inicio <= fecha_actual < hora_tardanza_fin:
            return "TARDANZA"
        else:
            return "FALTA"

    def save_attendance(self,
                        user_group_id: str,
                        user_name: str,
                        confidence: float,
                        tipo_asistencia: Optional[str] = None) -> Dict:
        """
```

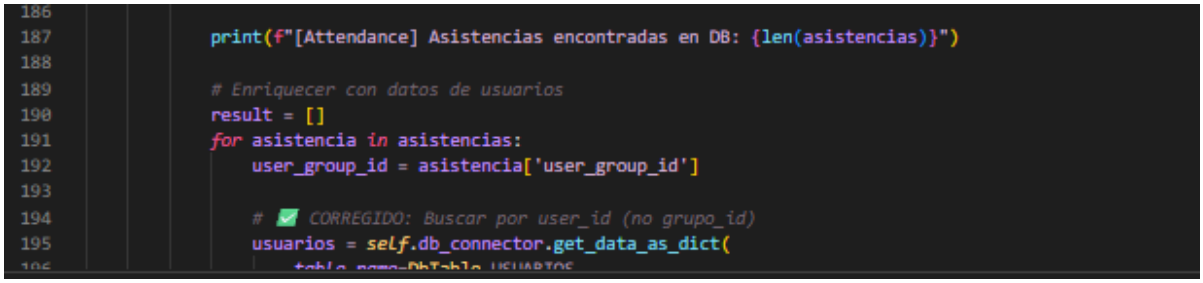
```python
    def save_attendance(self,
                        tipo_asistencia: Optional[str] = None) -> Dict:
        """
        Guarda un registro de asistencia.

        Args:
            user_group_id: ID del grupo del usuario
            user_name: Nombre del usuario
            confidence: Confianza del reconocimiento (0-1)
            tipo_asistencia: Opcional. Si no se proporciona, se calcula automáticamente

        Returns:
            Diccionario con resultado
        """
        try:
            # Obtener fecha y hora actual
            fecha_actual = datetime.now()
            fecha_str = fecha_actual.strftime("%d-%m-%Y")
            hora_str = fecha_actual.strftime("%H:%M:%S")

            # Determinar tipo de asistencia si no se proporciona
            if tipo_asistencia is None:
                tipo_asistencia = self.validate_attendance_state(fecha_actual)

            # Obtener modelo actual
            current_model = model_manager.get_current_model()
            modelo_id = current_model.model_id if current_model else 1

            # ✅ NUEVO: Obtener grupo_id del usuario desde la BD
            usuario_data = self.db_connector.get_data_as_dict(
                table_name=DbTable.USUARIOS,
                condition_dict={'user_id': user_group_id}
            )

            grupo_id = None
```

```python
            if usuario_data:
                grupo_id = usuario_data[0].get('grupo_id')

            # Verificar si ya existe asistencia para hoy
            existing = self.db_connector.get_data(
                table_name=DbTable.DETALLE_ASISTENCIA,
                condition_dict={
                    'user_group_id': user_group_id,
                    'fecha': fecha_str,
                    'modelo_id': modelo_id
                }
            )

            if existing:
                print(f"[Attendance] ⚠️ {user_name} ya tiene asistencia registrada hoy")
                return {
                    'success': False,
                    'message': f'{user_name} ya tiene asistencia registrada',
                    'already_exists': True,
                    'existing_record': {
                        'fecha': fecha_str,
                        'tipo': existing[0][3] if len(existing[0]) > 3 else 'Unknown'
                    }
                }
```

```python
class AttendanceService:
    def save_attendance(self,
                        }

                # Guardar nueva asistencia
                asistencia_id = str(uuid.uuid4())

                self.db_connector.create_or_update(
                    table_name=DbTable.DETALLE_ASISTENCIA,
                    condition_dict={'idasistencia': asistencia_id},
                    data={
                        'user_group_id': user_group_id,
                        'modelo_id': modelo_id,
                        'grupo_id': grupo_id,  # ✅ NUEVO: Guardar grupo_id
                        'tipo_asistencia': tipo_asistencia,
                        'fecha': fecha_str,
                        'hora': hora_str,
                        'confianza': confidence
                    }
                )

                print(f"[Attendance] ✅ Asistencia guardada: {user_name} - {tipo_asistencia} (Grupo: {grupo_id})")

                return {
                    'success': True,
                    'message': f'Asistencia de {user_name} guardada',
                    'asistencia_id': asistencia_id,
                    'tipo': tipo_asistencia,
                    'fecha': fecha_str,
                    'hora': hora_str,
                    'confidence': confidence,
                    'grupo_id': grupo_id
                }

            except Exception as e:
```

core > attendance_service.py > ...

```python
class AttendanceService:
    def save_attendance(self,

            except Exception as e:
                print(f"[Attendance] ❌ Error al guardar asistencia: {e}")
                return {
                    'success': False,
                    'message': f'Error al guardar asistencia: {str(e)}'
                }

    def get_attendance_by_date(self, fecha: str, modelo_id: Optional[int] = None) -> List[Dict]:
        """
        Obtiene asistencias de una fecha especifica.

        Args:
            fecha: Fecha en formato "dd-mm-yyyy"
            modelo_id: Opcional. ID del modelo (None = modelo actual)

        Returns:
            Lista de asistencias
        """
        try:
            # Si no se especifica modelo, usar el actual
            if modelo_id is None:
                current_model = model_manager.get_current_model()
                modelo_id = current_model.model_id if current_model else 1

            print(f"[Attendance] Consultando asistencias: fecha={fecha}, modelo={modelo_id}")
```



```
        print(f"[Attendance] Asistencias encontradas en DB: {len(asistencias)}")

        # Enriquecer con datos de usuarios
        result = []
        for asistencia in asistencias:
            user_group_id = asistencia['user_group_id']

            # ✅ CORREGIDO: Buscar por user_id (no grupo_id)
            usuarios = self.db_connector.get_data_as_dict(
                table_name=DbTable.USUARIOS,
                condition_dict={'user_id': user_group_id}
            )

            if usuarios:
                usuario = usuarios[0]
                result.append({
                    'id': asistencia['idasistencia'],
                    'user_university_code': usuario.get('codigo_universitario', ''),
                    'user_fullname': usuario.get('nombre_completo', 'Unknown'),
                    'type_assintance': asistencia['tipo_asistencia'],
                    'hora': asistencia.get('hora', ''),
                    'confianza': asistencia.get('confianza', 0.0),
                    'modelo_id': asistencia.get('modelo_id', 0)
                })
            else:
                print(f"[Attendance] ⚠️ Usuario no encontrado para user_id={user_group_id}")

        print(f"[Attendance] ✅ Obtenidas {len(result)} asistencias con datos completos")
        return result

    except Exception as e:
        print(f"[Attendance] ❌ Error al obtener asistencias: {e}")
        return []

def update_attendance_batch(self, updates: List[Dict]) -> Dict:
    """
    Actualiza múltiples asistencias.

    Args:
        updates: Lista de diccionarios con:
            - user_university_code: Código del usuario
            - type_assintance: Nuevo tipo de asistencia

    Returns:
        Diccionario con resultado
    """
    try:
```

```python
class AttendanceService:
    def update_attendance_batch(self, updates: List[Dict]) -> Dict:
            fecha_actual = datetime.now().strftime("%d-%m-%Y")
            updated_count = 0

            # Obtener modelo actual
            current_model = model_manager.get_current_model()
            modelo_id = current_model.model_id if current_model else 1

            for update in updates:
                # Obtener usuario por código
                usuarios = self.db_connector.get_data_as_dict(
                    table_name=DbTable.USUARIOS,
                    condition_dict={'codigo_universitario': update['user_university_code']}
                )

                if not usuarios:
                    print(f"[Attendance] ⚠️ Usuario no encontrado: {update['user_university_code']}")
                    continue

                user_group_id = usuarios[0]['group_id']

                # Actualizar o crear asistencia
                self.db_connector.create_or_update(
                    table_name=DbTable.DETALLE_ASISTENCIA,
                    condition_dict={
                        'user_group_id': user_group_id,
                        'fecha': fecha_actual,
                        'modelo_id': modelo_id
                    },
                    data={
                        'tipo_asistencia': update['type_assintance'],
                        'hora': datetime.now().strftime("%H:%M:%S"),
                        'confianza': 1.0  # Manual = 100% confianza
                    }
                )

                updated_count += 1

            print(f"[Attendance] ✅ {updated_count} asistencias actualizadas")

            return {
                'success': True,
                'message': f'{updated_count} asistencias actualizadas',
```

```python
                'message': f'{updated_count} asistencias actualizadas',
                'updated_count': updated_count
            }

        except Exception as e:
            print(f"[Attendance] ❌ Error al actualizar asistencias: {e}")
            return {
                'success': False,
                'message': f'Error al actualizar asistencias: {str(e)}'
            }


if __name__ == '__main__':
    print("=== ATTENDANCE SERVICE ===")
    print("Para usar, importa desde tu API o main.py")
```

- ***Link de repositorio del proyecto completo en GitHub:***

  https://github.com/MaikolDX/proyecto-tesis-def.git