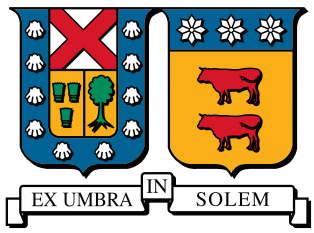
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAÍSO – CHILE



UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

SISTEMA DE REGISTRO BIOMÉTRICO DE ASISTENCIA MEDIANTE HUELLA DACTILAR Y ACCESO VÍA WEB

BRIAN JOHN ROBERTS VALENZUELA

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE
INGENIERÍA CIVIL ELECTRÓNICA

PROFESOR GUÍA: **AGUSTÍN GONZÁLEZ VALENZUELA**

PROFESOR COREFERENTE: **RUDY MALONNEK WETZEL**

DICIEMBRE - 2025

UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título ☐ Tesis de Postgrado

Título del trabajo: Sistema de registro biométrico de asistencia mediante huella dactilar y acceso vía web

Nombre del candidato(a): Brian John Roberts Valenzuela

Carrera / Grado: Ingeniería Civil Electrónica

Campus: Casa Central Valparaíso Departamento: Electrónica

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Agustín González Valenzuela, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL (marcar una opción)

☒ El trabajo **NO** contiene información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (embargo) por (marcar una opción):

☐ 6 meses ☐ 12 meses ☐ 2 años ☐ 3 años ☐ 5 años ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 12 / 03 / 2026 Firma: Agustín J. González

Estudiante o Candidato(a):

Fecha: 10/03/2026 Firma:

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

Agradecimientos

Antes de comenzar me gustaría agradecer profundamente a mi familia, quienes han sido siempre mi pilar y apoyo en cada etapa de mi vida académica y personal. A mis padres Ximena Valenzuela y John Roberts, por su incondicional apoyo a pesar de las dificultades que tuve en la Universidad. A mis hermanos Ian y Jeremy, por estar siempre presentes y acompañarme en los momentos difíciles y alegres.

Quiero también dedicar unas palabras a mi polola Josefina Vera, quien ha sido el atlas de mi mundo, sosteniéndome, apoyándome (aunque haga magia) y acompañándome con paciencia y cariño incondicional. Asimismo, a mi grupo de amigos más cercanos, Nico, Manu, Jose, Chato, Duki, Andros, Cutin, Pipe, Fido y Andaur, con quienes compartí risas, desvelos, conversaciones y momentos inolvidables que han hecho más llevadero este camino.

Finalmente, no puedo dejar de agradecer a todas las personas de la carrera y de la oficina de robótica, quienes contribuyeron no solo a mi formación académica, sino también a mi crecimiento personal. Agradezco los aprendizajes, la compañía y el ambiente de colaboración que hicieron de este proceso una experiencia enriquecedora, y por sobre todo a Fabian Clavijo, de Electrónica y compañero de la oficina, cuyas conversaciones, apoyo, risas y opiniones hicieron de este proceso uno muy grato.

Resumen

Sistema de registro biométrico de asistencia mediante huella dactilar y acceso vía web

Brian John Roberts Valenzuela

Memoria para optar al título de Ingeniero Civil Electrónico.

Profesor Guía: Agustín González Valenzuela

Diciembre 2025

Resumen

En muchas instituciones de educación superior el control de asistencia sigue realizándose de forma manual, ya sea mediante planillas en papel o registros digitales poco integrados con los sistemas académicos. Estos mecanismos son susceptibles a errores de registro, pérdida de información y, en algunos casos, permiten que un estudiante marque presencia por otro. Frente a este escenario, la biometría ofrece una alternativa más confiable para verificar la identidad de los alumnos y automatizar el proceso de toma de asistencia.

En esta memoria se diseña e implementa *BioAsist*, un sistema de registro de asistencia basado en huella dactilar compuesto por un dispositivo embebido portátil y una plataforma web de gestión. El dispositivo se implementa sobre un módulo ESP32 que integra un sensor de huella óptico R307, un display LCD y alimentación mediante *powerbank*, permitiendo su uso autónomo en sala. El firmware captura y valida huellas localmente y se comunica con un servidor mediante conexión WiFi, utilizando solicitudes HTTP para consultar estados de asistencia, ejecutar procesos de enrolamiento y registrar marcajes en una base de datos central.

La plataforma web, desarrollada en PHP con una base de datos MySQL, ofrece perfiles diferenciados para administradores, docentes y estudiantes. A través de ella se gestionan usuarios y ramos, se activan sesiones de asistencia y se visualizan reportes descargables en formato compatible con los sistemas académicos existentes. Las pruebas realizadas muestran que el prototipo permite registrar la asistencia de un curso de tamaño medio en tiempos adecuados, con una tasa de reconocimiento satisfactoria y una operación estable de la comunicación entre el dispositivo y el servidor.

Palabras clave: biometría, huella dactilar, control de asistencia, sistemas embebidos, ESP32, API REST.

Abstract

Biometric Attendance Registration System Using Fingerprint Recognition and Web Access

Brian John Roberts Valenzuela

Thesis submitted to obtain the degree of Electronic Civil Engineer.

Advisor: Agustín González Valenzuela

December 2025

Abstract

In many higher education institutions, attendance control is still carried out manually, either through paper forms or digital tools that lack proper integration with academic systems. These methods are prone to recording errors, data loss, and, in some cases, allow students to register attendance on behalf of others. In this context, biometric technologies offer a more reliable alternative for verifying student identity and automating the attendance-taking process.

This thesis presents the design and implementation of *BioAsist*, a fingerprint-based attendance registration system composed of a portable embedded device and a web management platform. The device is built around an ESP32 module integrating an R307 optical fingerprint sensor, an LCD display, and *powerbank*-based power supply, enabling autonomous operation in the classroom. The firmware captures and validates fingerprints locally and communicates with a server over WiFi using HTTP requests to query attendance states, execute enrollment procedures, and store attendance records in a central database.

The web platform, developed in PHP with a MySQL database, provides differentiated access for administrators, instructors, and students. Through this interface, users can manage courses, activate attendance sessions, and download reports compatible with existing academic systems. Experimental results show that the prototype can reliably register the attendance of a medium-sized class within acceptable time constraints, achieving a satisfactory recognition rate and stable communication between the embedded device and the server.

Keywords: biometrics, fingerprint recognition, attendance control, embedded systems, ESP32, REST API.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En el desarrollo de este capitulo se presenta el contexto general del proyecto, abordando la problemática que motiva el desarrollo de un sistema automatizado de control de asistencia y los objetivos que orientan su implementación. Se discuten los fundamentos y la relevancia de incorporar tecnologías biométricas en entornos educativos, así como el impacto que su adopción puede tener en la gestión académica y en la optimización de los procesos administrativos dentro de las instituciones de educación superior.

1.1. Contexto y motivación

El registro de asistencia constituye una actividad cotidiana en los procesos educativos, tanto a nivel escolar como universitario. Tradicionalmente, esta tarea se ha realizado de forma manual, ya sea mediante listas de papel que pasan entre los estudiantes o por el llamado de lista efectuado por el docente al inicio de la clase. En muchas asignaturas la asistencia representa un componente de evaluación o un requisito obligatorio, especialmente en los primeros ciclos formativos por lo que su control resulta fundamental para el cumplimiento académico. Sin embargo, los métodos tradicionales presentan limitaciones relevantes: consumen tiempo de la clase, son propensos a errores humanos y pueden ser objeto de fraude, como cuando un estudiante firma por otro compañero ausente..

En un escenario académico donde la digitalización de procesos ha adquirido un rol protagónico, resulta necesario modernizar también la gestión de la asistencia. La automatización de este procedimiento no solo busca reducir la carga administrativa de los docentes, sino también garantizar la confiabilidad de los registros y facilitar su integración con plataformas académicas. La biometría, en particular la huella dactilar, ofrece una solución práctica y de bajo costo para estos fines, dado que proporciona un mecanismo de identificación único, rápido y seguro, que no requiere portar credenciales físicas ni recordar contraseñas.

El presente proyecto se enmarca en esta necesidad, proponiendo el diseño e implementación de *BioAsist*, un sistema de registro de asistencia mediante huella dactilar que combina un *BioAsist-Device*, encargado de la captura y verificación biométrica, con una plataforma web y una base de datos central para la gestión de la información. De esta forma, se busca disponer de una herramienta confiable y eficiente, que además de optimizar el tiempo en sala, entregue valor agregado mediante reportes de asistencia filtrados por rangos de tiempo y opciones de exportación.

1.2. Problemática

El control manual de asistencia presenta limitaciones que afectan tanto a estudiantes como a docentes:

- Los registros en papel son poco confiables, dado que se pueden alterar y no siempre reflejan la asistencia real.
- El tiempo destinado a pasar lista reduce el tiempo efectivo de clase.
- La información registrada manualmente es difícil de sistematizar, lo que complica la generación de reportes consolidados.

Frente a estas dificultades, surge la necesidad de un sistema automatizado que garantice la autenticidad del registro y simplifique la administración de la información. La biometría, y en particular el uso de huellas dactilares, se presenta como una alternativa sólida que asegura la identificación única de cada estudiante y evita prácticas fraudulentas. BioAsist fue concebido precisamente con este propósito.

1.3. Objetivos

Objetivo general

Diseñar e implementar un sistema de registro biométrico de asistencia mediante huella dactilar, integrando un lector con un microcontrolador y un servidor web que centralice la información para su consulta en tiempo real.

Objetivos específicos

1. Configurar un *BioAsist-Device* integrado con microcontrolador para enrolamiento de usuarios, captura y verificación de huellas dactilares.
2. Almacenar de manera local las huellas capturadas en el *BioAsist-Device*, es decir, manteniéndolas de forma provisional hasta su sincronización con el servidor.
3. Transmitir los registros de asistencia a una base de datos central mediante conexión WiFi, incorporando un mecanismo de sincronización controlada por el usuario que asegure la coherencia de la información entre el dispositivo y el servidor, y preserve la integridad de los datos durante la transferencia, dado que cualquier inconsistencia o pérdida de información puede afectar directamente la validez de los registros académicos.
4. Desarrollar página web que permita a los usuarios consultar registros de asistencia en tiempo real y según su perfil con autenticación para el profesorado y estudiantes.

1.4. Metodología y enfoque

La metodología seguida en este proyecto consistió en un proceso iterativo de análisis, diseño, implementación y validación. Los principales pasos fueron los siguientes:

- **Selección tecnológica:** se evaluaron distintas alternativas de diseño, considerando criterios como escalabilidad, simplicidad de implementación, costo, robustez, acceso a datos y autonomía. Tras el análisis comparativo, se seleccionó un *sistema híbrido con sincronización periódica*, entendiendo por ello una arquitectura que combina almacenamiento y procesamiento local en el *BioAsist-Device* con una base de datos central en el servidor, permitiendo la actualización de la información en intervalos definidos o cuando exista conectividad disponible. Esta modalidad ofrece un equilibrio entre robustez y flexibilidad de uso en entornos con conectividad variable.

- **Implementación de hardware y firmware:** se definió el microcontrolador ESP32 [3] como plataforma principal, debido a su conectividad WiFi integrada, su capacidad de procesamiento y su soporte en entornos de desarrollo accesibles. Este se integró con un lector de huellas R307 [2] y una pantalla LCD para conformar el *BioAsist-Device*, encargado de la captura biométrica y la interacción básica con el usuario.

- **Desarrollo de software servidor:** se implementó un servidor web con base de datos MySQL para la gestión centralizada de los registros, y una interfaz web en PHP que permite visualizar y exportar los datos. Este componente, junto con el *BioAsist-Device*, conforma el sistema completo *BioAsist*.

- **Pruebas de desempeño:** se realizaron mediciones en cursos de tamaño medio para validar el tiempo de registro, la estabilidad de la comunicación con el servidor y la confiabilidad en la identificación biométrica.

Este enfoque permitió desarrollar un prototipo funcional que satisface los objetivos planteados y constituye una base sólida para su futura mejora y escalabilidad.

Capítulo 2

Tecnologías en sistemas de control de asistencia

Este capítulo presenta un análisis de las principales tecnologías utilizadas en el control de asistencia, desde los métodos tradicionales hasta los sistemas biométricos actuales. Asimismo, se profundiza en el uso de lectores de huella dactilar, los microcontroladores como plataformas de procesamiento, y las soluciones comerciales y académicas que se encuentran disponibles en la actualidad.

A lo largo del tiempo, los métodos utilizados para registrar asistencia han experimentado una evolución significativa, tanto en entornos educativos como laborales. En un comienzo, el control se realizaba de manera completamente manual, mediante firmas o listas en papel. Posteriormente, con la masificación de los sistemas informáticos, surgieron los primeros mecanismos automatizados basados en tarjetas magnéticas y códigos de barras, los cuales permitieron reducir errores humanos y agilizar el proceso de registro. Con el avance de la tecnología, se incorporaron soluciones de identificación por radiofrecuencia (RFID) y, más recientemente, sistemas biométricos que emplean características únicas del individuo, como la huella dactilar o el reconocimiento facial. Esta evolución refleja una tendencia hacia la automatización, la precisión y la seguridad en el registro de asistencia, buscando minimizar la intervención humana y garantizar la autenticidad de los datos recolectados.

2.1. Métodos tradicionales

El control de asistencia ha formado parte de los procesos administrativos en instituciones educativas durante décadas. Los métodos tradicionales más comunes incluyen:

- **Registro manual en papel:** el docente anota la asistencia de los estudiantes en una lista física. Si bien es un método simple, es altamente vulnerable a errores humanos, suplantaciones (estudiantes que responden por compañeros ausentes) y pérdida de registros.
- **Tarjetas magnéticas y códigos de barras:** requieren que cada alumno porte una credencial para ser escaneada. Mejoran la rapidez del proceso, pero introducen riesgos de extravío o préstamo de tarjetas, lo que afecta la confiabilidad.
- **Tecnologías de proximidad (RFID/NFC):** permiten registrar la asistencia acercando la tarjeta o dispositivo a un lector. Aunque son más rápidas y reducen el contacto físico, no evitan totalmente la suplantación de identidad si las credenciales se comparten.

Estos métodos, si bien aún son utilizados en muchos contextos, no garantizan plenamente la autenticidad del registro. Esto ha impulsado el avance hacia sistemas de identificación biométrica.

2.2. Sistemas biométricos

La biometría corresponde al conjunto de técnicas que permiten identificar o autenticar a una persona a partir de características físicas o de comportamiento que son únicas, medibles y relativamente permanentes en el tiempo [4]. Entre las características biométricas más utilizadas se encuentran:

- **Huellas dactilares:** basadas en el patrón único de crestas y valles de los dedos.
- **Reconocimiento facial:** analiza rasgos faciales mediante imágenes o video.
- **Reconocimiento de iris:** utiliza la textura única del iris en los ojos.
- **Reconocimiento de voz:** analiza patrones vocales individuales.
- **Firma biométrica o dinámica de escritura:** considera trazos, velocidad y presión al escribir.

Los sistemas biométricos presentan ventajas claras frente a métodos tradicionales: no requieren portar objetos externos ni recordar contraseñas, reducen la posibilidad de suplantación y permiten un registro más eficiente. En la actualidad, el reconocimiento facial y las huellas dactilares son los métodos biométricos más difundidos en entornos de control de acceso y asistencia, mientras que el reconocimiento de iris y voz se emplean en aplicaciones más específicas y de mayor seguridad [4].

2.3. Lector de huella dactilar

La huella dactilar es uno de los identificadores biométricos más utilizados en sistemas de control de asistencia debido a su unicidad, estabilidad a lo largo de la vida y facilidad de captura. A diferencia del reconocimiento facial o de iris, el costo de implementación es significativamente menor, y la tecnología está ampliamente disponible en dispositivos portátiles y de bajo consumo energético.

Existen principalmente dos tipos de lectores de huella dactilar:

- **Lectores ópticos:** utilizan un sensor basado en cámaras o fotodiodos que capturan una imagen de la huella mediante reflexión de la luz. Son confiables y económicos, pero pueden ser sensibles a la suciedad, humedad o a intentos de suplantación con imágenes [5]. La Figura 2.1 ilustra el principio de funcionamiento de un lector de huella óptico, donde un prisma y un sensor de imagen permiten capturar los relieves de la superficie dactilar mediante reflexión de luz.
- **Lectores capacitivos:** miden las variaciones de capacitancia eléctrica producidas por las crestas y valles de la huella. Son más resistentes a falsificaciones y no requieren condiciones de iluminación específicas, aunque suelen tener un costo mayor [6].

En el caso de este proyecto, se optó por un lector **óptico** (modelo R307), el cual, en conjunto con el ESP32 y una pantalla LCD, conforma el *BioAsist-Device*. Este dispositivo fue seleccionado debido a su disponibilidad en el mercado, buena documentación técnica, bajo costo y facilidad

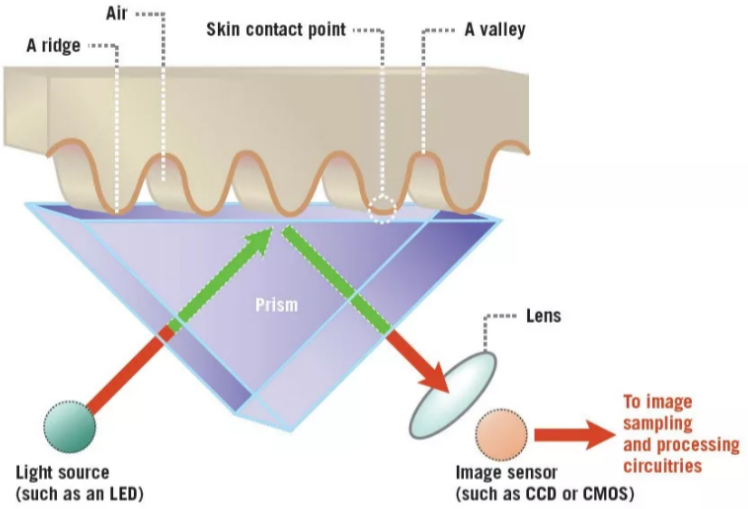


Figura 2.1: Ejemplo de lector de huella óptico. El sistema utiliza un *optical sensor* basado en reflexión de luz para capturar la imagen de la huella. Fuente: Dorlo Technologies, "Optical sensor vs. capacitive sensor" (s.f.). [1]

de integración con microcontroladores. Estas características lo hacen adecuado para un sistema académico de asistencia como *BioAsist*, donde se requiere confiabilidad aceptable y costos accesibles.

2.4. Microcontroladores y comunicación con servidores

Los microcontroladores constituyen el núcleo de los sistemas embebidos, integrando en un solo chip capacidades de cómputo, memoria y comunicación [3]. En el ámbito de control de asistencia, cumplen las siguientes funciones:

- Capturar y procesar datos del lector biométrico.
- Ejecutar algoritmos de validación y comunicación con el servidor central del sistema *BioAsist*.
- Enviar registros al servidor encargado de administrar la base de datos y procesar las solicitudes desde los dispositivos, mediante protocolos de red.

En este proyecto, el microcontrolador **ESP32** fue la plataforma seleccionada por las siguientes ventajas frente a alternativas como Arduino UNO o PIC tradicionales:

- Incorpora conectividad Wi-Fi y Bluetooth de manera nativa.
- Posee mayor capacidad de procesamiento y memoria.
- Cuenta con amplia comunidad y soporte de librerías en entornos de desarrollo como Arduino IDE.

El ESP32, junto con el lector R307 y la pantalla LCD, conforma el *BioAsist-Device*. Este dispositivo es responsable de la captura biométrica y la comunicación con el servidor central, que

forma parte del sistema *BioAsist* en su conjunto. De esta manera, el sistema integra el hardware embebido con la infraestructura web para entregar un servicio de registro y consulta de asistencia en tiempo real.

2.5. Soluciones tecnológicas actuales en control de asistencia biométrica

En el mercado existen múltiples soluciones que integran biometría para el control de asistencia, las cuales pueden clasificarse según su ámbito de implementación y propósito de uso:

- **Dispositivos comerciales:** terminales de asistencia autónomos que integran huella dactilar, reconocimiento facial o tarjetas inteligentes. Estos sistemas operan de forma local en el punto de control, son robustos, pero suelen tener costos elevados y menor flexibilidad de personalización.
- **Sistemas en la nube:** soluciones orientadas a dispositivos móviles o plataformas web que utilizan cámaras o sensores integrados para el reconocimiento facial o dactilar, con procesamiento remoto y sincronización inmediata en servidores externos.
- **Proyectos académicos y de bajo costo:** diversos estudios han mostrado la viabilidad de integrar sensores de huella con microcontroladores y servidores web para aplicaciones educativas, destacando el equilibrio entre costo y efectividad [7].

El análisis de estas alternativas muestra que la huella dactilar es un estándar ampliamente adoptado, y que soluciones basadas en microcontroladores como el *BioAsist-Device* permiten desarrollar sistemas personalizados, escalables y de bajo costo. En este contexto, *BioAsist* se posiciona como una propuesta factible y aplicable en entornos académicos.

Capítulo 3

Análisis del problema y requerimientos de BioAssist

El presente capítulo tiene por objetivo analizar el problema que se busca resolver, traducirlo en requerimientos claros para el sistema y establecer los lineamientos técnicos y de seguridad que guiarán su desarrollo. Para ello, se identifican los requisitos funcionales y no funcionales, los usuarios del sistema, las necesidades de hardware y software, y finalmente se abordan las consideraciones de seguridad y privacidad de la información.

3.1. Requisitos funcionales

Los requisitos funcionales definen las capacidades que *BioAsist* debe proporcionar al usuario final para cumplir con el propósito de control de asistencia. Entre los principales se encuentran:

- **Registro de asistencia mediante huella digital:** el sistema debe permitir que cada estudiante registre su asistencia colocando su dedo sobre el *BioAsist-Device*.
- **Validación en tiempo real:** la verificación de la huella debe realizarse de forma inmediata, entregando retroalimentación al usuario (presente/ausente).
- **Visualización de asistencia:** los docentes deben contar con una interfaz que les permita ver en tiempo real la asistencia de la clase en curso, así como generar reportes por día, semana o mes. Asimismo, el sistema debe permitir que los estudiantes consulten su propio registro de asistencia de manera individual.
- **Exportación de registros:** el sistema debe generar archivos en formato CSV u otro compatible con Moodle, para facilitar la integración con plataformas académicas.
- **Administración de sesiones de asistencia:** el profesor debe poder iniciar y finalizar el registro de asistencia de manera controlada.

3.2. Requisitos no funcionales

Los requisitos no funcionales establecen las condiciones de calidad, rendimiento y confiabilidad que debe cumplir *BioAsist*:

- **Disponibilidad:** el sistema debe permanecer operativo durante toda la jornada académica, garantizando la continuidad del servicio incluso ante reconexiones o reinicios del dispositivo.
- **Rendimiento:** el tiempo de procesamiento de una lectura de huella dactilar no debe superar los 2 segundos por estudiante, considerando la verificación local y la confirmación visual en el dispositivo.
- **Escalabilidad:** la arquitectura debe permitir la incorporación de nuevos dispositivos *BioAsist-Device* o cursos adicionales sin afectar el desempeño general del sistema.
- **Usabilidad:** la interfaz de usuario debe ser intuitiva y proporcionar retroalimentación clara tanto al profesorado como a los estudiantes durante el registro y la consulta de asistencia.
- **Confiabilidad:** el reconocimiento de huella debe alcanzar una tasa de éxito superior al 99 %, de acuerdo con las especificaciones técnicas del sensor R307 (FAR $<$ 0,001 %, FRR $<$ 0,1 %) [2].[1]

3.3. Identificación de usuarios

BioAsist contempla distintos perfiles de usuario con responsabilidades diferenciadas:

- **Estudiantes:** registran su asistencia mediante el *BioAsist-Device*.
- **Docentes:** gestionan el inicio y cierre de sesiones de asistencia, y acceden a los reportes de asistencia de sus cursos.
- **Administradores:** gestionan las cuentas de usuario y la configuración general del sistema.

3.4. Requisitos de hardware y software

Hardware

- **BioAsist-Device:** conformado por un ESP32 DevKit V1 [3], encargado de la comunicación con el sensor biométrico y la conexión Wi-Fi, un lector de huellas R307 [2] para la captura y validación local, y una pantalla LCD 16x2 para entregar retroalimentación básica.
- **Servidor:** equipo de cómputo con capacidad de ejecutar servicios PHP y MySQL, encargado de centralizar la información y disponibilizarla a los usuarios a través de la web.

Software

- **Lenguaje de programación embebido:** Arduino (C/C++) para la lógica del *BioAsist-Device*, elegido por su amplia compatibilidad con la familia ESP32, disponibilidad de librerías para sensores biométricos y facilidad de integración con servicios web.
- **Servidor web:** Apache, ejecutado mediante el paquete XAMPP, encargado de procesar los archivos PHP y servir las vistas HTML. Se optó por Apache debido a su estabilidad, carácter de software libre y compatibilidad con PHP y MySQL.

[1] El *False Acceptance Rate* (FAR) indica la probabilidad de aceptar una huella incorrecta, y el *False Rejection Rate* (FRR) la probabilidad de rechazar una huella válida.

- **Gestor de base de datos:** MySQL, utilizado para almacenar usuarios y registros de asistencia. Se seleccionó por su integración nativa con PHP, bajo consumo de recursos y facilidad para realizar consultas estructuradas.
- **Lenguaje de servidor:** PHP, empleado para la lógica del servidor y la generación dinámica de las vistas web. Se eligió por su amplia disponibilidad en servidores Apache, su simplicidad para gestionar formularios y solicitudes HTTP, y su integración directa con bases de datos MySQL.
- **Plataforma académica:** integración con Moodle mediante exportación de archivos CSV compatibles con su módulo de asistencia. Esta característica permite que los registros capturados por el sistema *BioAsist* puedan ser importados directamente en cursos institucionales, facilitando la gestión académica y reduciendo tareas manuales.

3.5. Consideraciones de seguridad y privacidad

Dado que *BioAsist* trabaja con información biométrica y datos personales de los estudiantes, es esencial aplicar medidas de seguridad y resguardar la privacidad:

- **Protección de datos biométricos:** las plantillas de huellas no deben almacenarse en texto plano, sino en formato encriptado en la memoria del *BioAsist-Device* o del servidor.
- **Control de accesos:** los perfiles de administrador y docente deben autenticarse antes de acceder al sistema.
- **Confidencialidad de registros:** los reportes de asistencia deben estar protegidos mediante control de acceso, de modo que sólo el profesorado y los administradores puedan consultar la información completa de sus cursos. Los estudiantes, en cambio, podrán acceder únicamente a su propio historial de asistencia para cada una de las asignaturas que este cursando.

Capítulo 4

Diseño de *BioAsist*

Este capítulo presenta el diseño general del sistema *BioAsist*, abarcando la arquitectura de hardware y software, la estructura de comunicación y los módulos que lo componen. Se describen las decisiones técnicas adoptadas para garantizar la funcionalidad, escalabilidad y confiabilidad del sistema, así como la integración entre el dispositivo biométrico, el servidor y la plataforma web utilizada para la gestión de asistencia.

4.1. Arquitectura de *BioAsist*

El sistema propuesto fue diseñado con un enfoque modular, compuesto por un *BioAsist-Device* autónomo con capacidad de registro biométrico, un servidor basado en tecnologías web y una base de datos relacional. Esta arquitectura permite a los docentes gestionar de manera eficiente el proceso de toma de asistencia estudiantil a través de una interfaz intuitiva y desde cualquier navegador compatible en la red.

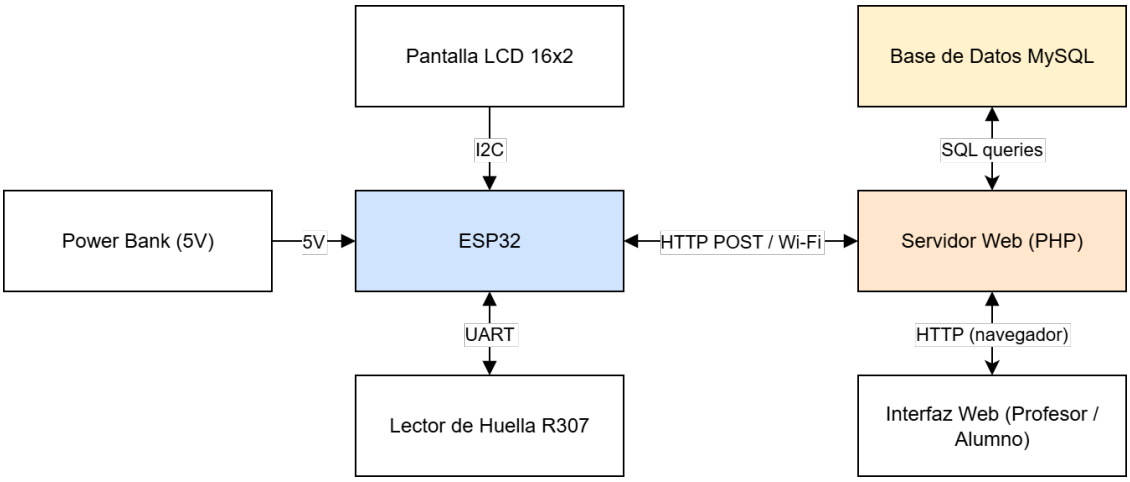


Figura 4.1: Diagrama de arquitectura general del sistema propuesto

La Figura 4.1 muestra una vista general de *BioAsist*, detallando los principales componentes y la interacción entre ellos. Estos son:

- **Power Bank:** Fuente de alimentación portátil encargada de proporcionar energía al microcontrolador ESP32 y a los periféricos conectados, permitiendo la operación autónoma y

móvil del *BioAsist-Device*.

- **ESP32:** Microcontrolador principal que coordina toda la lógica del *BioAsist-Device*. Se conecta a la red Wi-Fi local para consultar y actualizar los *endpoints* expuestos por el servidor web. Entre sus funciones se incluyen: enviar solicitudes HTTP (GET y POST), asignar de forma automática un `finger_id` al momento de enrolar una nueva huella, verificar huellas capturadas en tiempo real, y gestionar la interacción con el usuario a través de una pantalla LCD. El ESP32 actúa como intermediario inteligente entre el sensor biométrico y el servidor central.

  En el prototipo actual, el *BioAsist-Device* funciona como único enrolador del sistema, por lo que el `finger_id` generado localmente por el lector R307 resulta suficiente para identificar cada huella de forma unívoca dentro del dispositivo. Sin embargo, en un escenario con múltiples dispositivos enroladores, la asignación local de `finger_id` podría generar colisiones entre equipos distintos. Para abordar este desafío de escalabilidad, existen dos alternativas ampliamente utilizadas:

  - La asignación centralizada desde el servidor, en la que el *BioAsist-Device* solicita un identificador disponible antes de almacenar la plantilla, garantizando unicidad global.
  - El uso de una clave compuesta `(device_id, finger_id)`, permitiendo que cada dispositivo gestione sus propios índices locales mientras el servidor asegura la unicidad mediante esta combinación. Ambas estrategias son compatibles con el diseño propuesto y aseguran integridad y consistencia en sistemas con múltiples dispositivos.

- **Pantalla LCD 16x2:** Display utilizado para proporcionar retroalimentación visual al usuario durante los procesos de enrolamiento y toma de asistencia. Se comunica con el ESP32 mediante el protocolo I2C.

- **Lector de huellas:** Módulo biométrico que permite la captura, almacenamiento y comparación de huellas dactilares. Está conectado al ESP32 mediante UART y trabaja en conjunto con este para ejecutar rutinas de enrolamiento y verificación.

- **Servidor Web** Actúa como núcleo de *BioAsist*. Recibe solicitudes del *BioAsist-Device* vía HTTP y gestiona la base de datos relacional. Contiene endpoints específicos para enrolar huellas, registrar asistencias, actualizar estados y consultar registros.

- **Base de datos relacional:** Almacena la información estructurada de *BioAsist*: usuarios, ramos, asistencia, enrolamientos y estados. Su diseño fue normalizado hasta 3FN para garantizar integridad, coherencia y flexibilidad.

- **Interfaz web para docentes (HTML/CSS/JS):** Permite al docente gestionar sus ramos, iniciar o finalizar la toma de asistencia, inscribir alumnos, consultar registros y exportarlos. Esta interfaz accede a los datos del servidor mediante sesiones y consultas internas a la base de datos.

El detalle del diseño interno del *BioAsist-Device* y sus protocolos de comunicación se describe en las secciones 4.2 y 4.3 respectivamente.

4.2. Módulo de Captura de Huella: Diseño y Lógica

El módulo de captura de huella constituye el núcleo del proceso de identificación biométrica de *BioAsist*. Está compuesto por el sensor R307 conectado al microcontrolador ESP32 mediante una interfaz UART. Este módulo permite tanto el enrolamiento de nuevas huellas como la verificación de huellas ya registradas, habilitando así la trazabilidad y automatización del proceso de asistencia.

El R307 es un lector de huellas dactilares óptico ampliamente utilizado en sistemas embebidos. Su funcionalidad incluye la adquisición de imágenes, generación de plantillas, comparación y almacenamiento interno de hasta 1000 huellas. Ofrece una comunicación serial TTL (UART), donde *TTL* corresponde al estándar *Transistor–Transistor Logic*, que define los niveles lógicos utilizados en circuitería digital (0 V para un "0" lógico y entre 3.3–5 V para un "1" lógico) [8]. Este tipo de interfaz es directamente compatible con microcontroladores como el ESP32, a diferencia de la comunicación UART basada en RS-232, la cual emplea voltajes significativamente mayores (±7 a ±12 V) e incompatibles sin un conversor adicional como el MAX232. El módulo cuenta con comandos estándar para enrolar, verificar, buscar huellas y eliminar entradas.

En *BioAsist-Device*, el R307 permanece en escucha constante desde el ESP32, a la espera de comandos que se activan según el estado de la aplicación.

4.2.1. Flujo de enrolamiento implementado en el prototipo

El prototipo desarrollado de *BioAsist* opera con un único *BioAsist-Device* encargado del proceso completo de enrolamiento. En este escenario, el identificador biométrico (`finger_id`) puede ser asignado localmente por el lector R307 sin riesgo de colisiones entre dispositivos. El flujo implementado actualmente es el siguiente:

1. El servidor crea un registro en la tabla `enrolamientos` con estado *pendiente* cuando el docente habilita esta opción desde la interfaz web.
2. El *BioAsist-Device*, mediante sondeo periódico (polling), detecta el estado pendiente e inicia la rutina de enrolamiento, solicitando al usuario colocar el dedo dos veces de forma consecutiva.
3. El ESP32 genera una plantilla a partir de ambas capturas y, antes de almacenarla, ejecuta una rutina de mapeo que recorre las posiciones de memoria del R307 hasta encontrar un espacio libre. El índice de esa posición se utiliza como `finger_id`, que queda registrado tanto en la memoria interna del sensor como en la base de datos del servidor.
4. La plantilla se almacena en la posición correspondiente del sensor R307.
5. El dispositivo notifica al servidor mediante una solicitud HTTP POST que la huella fue enrolada exitosamente.
6. El servidor actualiza el registro en la base de datos, asociando el `finger_id` generado con el usuario correspondiente.

Este flujo es adecuado para el prototipo, pero presenta limitaciones cuando se requieren múltiples dispositivos, dado que la asignación local de `finger_id` puede generar colisiones entre equipos.

4.2.2. Flujo de enrolamiento en un sistema escalable

En un escenario de despliegue real, el sistema *BioAsist* debe ser capaz de operar con múltiples *BioAsist-Device* distribuidos en distintas salas o edificios. Bajo estas condiciones, la asignación local de `finger_id` deja de ser suficiente para garantizar unicidad global, ya que dos dispositivos distintos podrían almacenar una plantilla en la misma posición de memoria interna del sensor R307. Para abordar este desafío, la escalabilidad del sistema puede lograrse manteniendo la asignación local del `finger_id` en cada dispositivo, pero asegurando unicidad a nivel del sistema mediante la combinación `(device_id, finger_id)` en la base de datos.

El flujo propuesto para un sistema escalable es el siguiente:

1. El servidor crea un registro de enrolamiento con estado *pendiente* cuando el docente habilita esta opción desde la interfaz web.
2. Cada *BioAsist-Device* monitorea periódicamente este estado y, al detectarlo, solicita al usuario que coloque el dedo para capturar la huella.
3. Una vez generada la plantilla, el dispositivo ejecuta una rutina interna de mapeo que recorre las posiciones de memoria del R307 para determinar un `finger_id` disponible. Este identificador corresponde al índice físico donde se almacenará la plantilla dentro del sensor biométrico.
4. El dispositivo almacena la plantilla en la posición seleccionada del R307 y envía al servidor una notificación indicando que el proceso fue exitoso. Esta notificación incluye el par `(device_id, finger_id)` que identifica de manera única la huella en el sistema.
5. El servidor verifica que la combinación `(device_id, finger_id)` no se encuentre previamente registrada en la base de datos. En caso de detectar una colisión, el servidor puede solicitar al dispositivo la asignación de otro `finger_id` o marcar el enrolamiento como fallido.
6. Si no existen colisiones, el servidor actualiza la base de datos, asociando el par `(device_id, finger_id)` con el usuario enrolado y marcando el proceso como completado.

Este enfoque preserva la lógica interna del prototipo —donde el `finger_id` corresponde al índice local utilizado por el sensor R307— y a la vez garantiza unicidad global mediante su combinación con el `device_id`. De esta manera, es posible incorporar múltiples dispositivos sin generar conflictos, permitiendo que el sistema escale de forma consistente dentro de entornos institucionales.

4.2.3. Registrar asistencia

Cuando el estado de asistencia en la base de datos es activado desde la interfaz web, el ESP32 cambia su lógica a modo verificación. En este modo:

1. El R307 lee una huella colocada por el alumno.
2. El ESP32 compara la huella capturada con las almacenadas en el sensor.
3. Si se encuentra coincidencia, se consulta el `finger_id` y se identifica al alumno.
4. Se envía un POST al servidor para registrar la asistencia como presente.

5. Si no se detecta coincidencia tras varios intentos, el ESP32 muestra un mensaje de error.

En caso de fallos durante el enrolamiento (ej.: imágenes inconsistentes, dedo mal posicionado, huella ya existente, etc.), *BioAsist* notifica al docente mediante la interfaz web. También se implementó la opción de cancelar un enrolamiento desde el servidor, lo cual cambia el estado a `cancelado` y detiene la lógica en el ESP32.

La Figura 4.2 resume el flujo completo de operación del BioAsist-Device, integrando la inicialización, la consulta de estados al servidor, el enrolamiento y la verificación de huellas para el registro de asistencia.

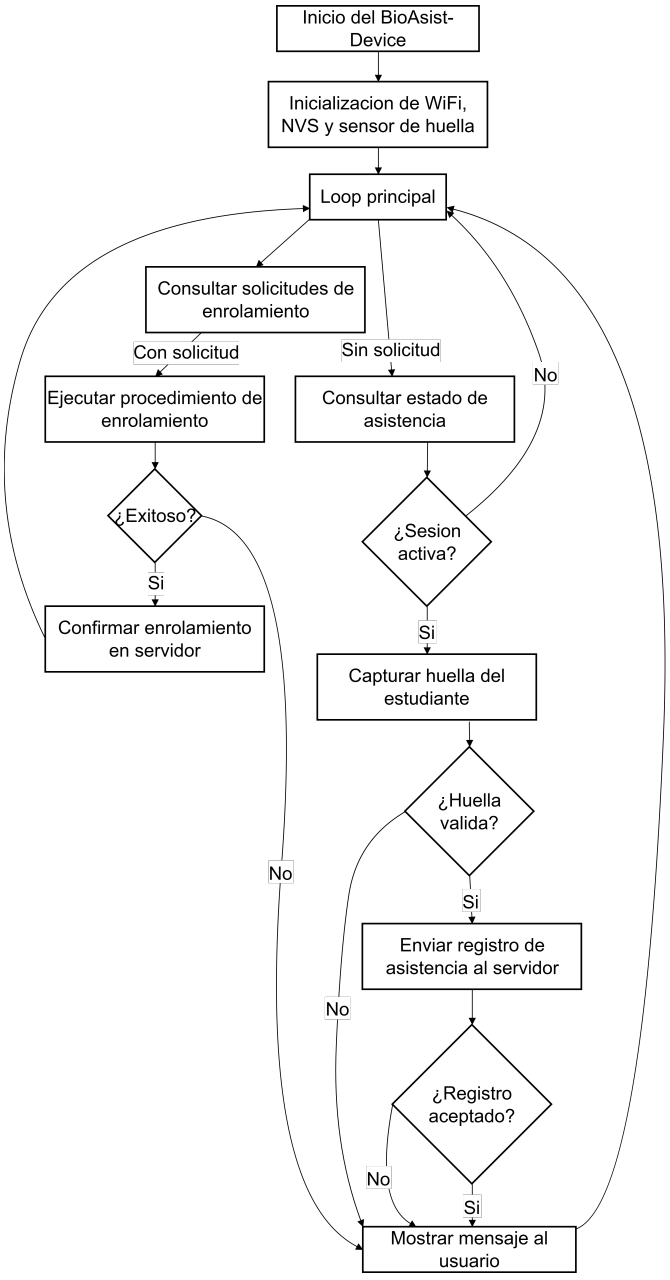


Figura 4.2: Flujo completo de operación del BioAsist-Device.

4.3. Comunicación con el servidor: protocolos y flujos

La comunicación entre el *BioAsist-Device* y el servidor web se basa en el protocolo HTTP, aprovechando las capacidades de conectividad Wi-Fi del microcontrolador. Esta interacción sigue un modelo cliente-servidor, donde el ESP32 actúa como cliente que realiza solicitudes a los *endpoints* definidos en el servidor, el cual responde con la información requerida o procesa las peticiones.

Dado el alcance del proyecto, la arquitectura ya existente del backend y el objetivo de mantener una implementación sencilla y compatible con entornos universitarios típicos, se optó por utilizar el protocolo HTTP en lugar de alternativas más complejas como MQTT o WebSockets.

MQTT es un protocolo de mensajería liviano orientado a escenarios IoT, basado en un esquema *publish/subscribe*, eficiente en redes inestables o de bajo ancho de banda. Funciona a través de un *broker* central que distribuye mensajes entre clientes [9].

Por otro lado, **WebSockets** permiten una comunicación bidireccional y persistente entre cliente y servidor, habilitando la transmisión continua de datos en tiempo real sin necesidad de realizar solicitudes repetitivas como en HTTP [10].

Si bien ambas tecnologías ofrecen ventajas para aplicaciones con alto dinamismo o telemetría en tiempo real, su incorporación habría aumentado la complejidad del sistema, tanto en la implementación del servidor como en el firmware del dispositivo. Considerando que el flujo de interacción entre el lector y el servidor consiste principalmente en solicitudes discretas y eventos puntuales (consultar estados y registrar asistencia), HTTP resulta suficiente y consistente con los requerimientos del proyecto.

Las solicitudes se realizan mediante los métodos GET y POST, según el tipo de operación:

- **GET**: Consultas periódicas al servidor para verificar estados. Por ejemplo, el *BioAsist-Device* consulta si debe iniciar una sesión de asistencia o si se ha solicitado el enrolamiento de una nueva huella.
- **POST**: Envío de información hacia el servidor. Esto incluye registrar una huella enrolada con su `finger_id`, o marcar la asistencia de un alumno validado.

La comunicación se realiza utilizando el formato *application/x-www-form-urlencoded* [11], donde los datos se envían como pares `clave=valor` en el cuerpo de la solicitud POST. Este formato, utilizado por los formularios HTML tradicionales, es interpretado de forma nativa por PHP a través de `$_POST`, por lo que el ESP32 puede generarlo directamente como una cadena de texto sin requerir mecanismos de codificación adicionales. Alternativamente, podría haberse utilizado JSON, pero no era necesario en este caso dado el bajo volumen de datos transmitidos y la compatibilidad con las funciones estándar de PHP para formularios.

El ESP32 mantiene un mecanismo de *polling* periódico (cada 2–3 segundos) para consultar el estado actual de las acciones solicitadas por el docente (enrolamiento o toma de asistencia). Este esquema permite que el dispositivo funcione de forma reactiva a través de su interfaz web, sin necesidad de mantener una conexión persistente.

El protocolo de comunicación entre el *BioAsist-Device* y el servidor puede esquematizarse de la siguiente forma:

Algorithm 1: Protocolo de comunicación entre *BioAsist-Device* y el servidor web

```
Función setup()
    configurarGPIO();
    conectarWiFi(red_configurada);

Función loop()
    estadoAsistencia = GET("/estado_asistencia.php");
    estadoEnrolamiento = GET("/estado_enrolamiento.php");
    if estadoEnrolamiento == "ENROLAR": then
        resultado = ejecutarEnrolamiento();
        POST("/resultado_enrolamiento.php", resultado);
    end
    if estadoAsistencia == "INICIAR": then
        resultado = procesarAsistencia();
        POST("/resultado_asistencia.php", resultado);
    end
    esperar(T_poll);
```

A partir de este pseudocódigo, la interacción se puede describir de la siguiente forma:

1. Al iniciar el programa (en `setup()`), el ESP32 configura los pines necesarios y se conecta a la red Wi-Fi previamente configurada.

2. En el ciclo principal, el dispositivo consulta periódicamente, mediante solicitudes GET, los endpoints `estado_asistencia.php` y `estado_enrolamiento.php`. Se reconoce que este mecanismo de consulta constante implica un mayor consumo energético en comparación con técnicas más eficientes, como notificaciones *push* o conexiones persistentes; sin embargo, resulta adecuado para este prototipo, ya que simplifica la arquitectura del sistema y mantiene un consumo aceptable al operar con una batería externa.

3. Si el servidor indica una acción pendiente, el dispositivo activa la lógica correspondiente, ya sea (i) iniciar un proceso de enrolamiento o (ii) procesar la verificación de una huella para registrar la asistencia de un alumno.

4. Una vez procesada la acción (enrolamiento o validación de asistencia), el ESP32 realiza una solicitud POST al servidor informando el resultado y los datos asociados (por ejemplo, el `finger_id` y el identificador del alumno).

5. El servidor procesa la solicitud y actualiza la base de datos MySQL, dejando registrado el estado de la operación (nuevo enrolamiento o marca de asistencia).

Se consideraron mecanismos básicos de manejo de errores como validación de respuestas, reintentos en caso de error de red y mensajes de estado en pantalla para notificar al usuario en caso de fallos de conexión.

En general, el diseño de comunicación implementado se ajustó adecuadamente a los requerimientos de BioAsist, incorporando y adaptando parte del código base proporcionado por la librería oficial del sensor R307 [12] para integrarlo con la lógica de comunicación HTTP y los servicios del servidor, lo que permitió una operación fluida.

4.4. Interfaz web del Sistema BioAsist

La interfaz web de *BioAsist* constituye el módulo central de gestión del sistema, permitiendo a los distintos perfiles de usuario acceder a las funciones que les corresponden según su rol. Si bien este capítulo se centra en el diseño general del sistema, en esta sección se incluyen también ejemplos visuales de la interfaz ya implementada, con el propósito de ilustrar cómo las decisiones de diseño se materializaron finalmente en la plataforma. La descripción detallada de la implementación técnica se desarrolla posteriormente en el Capítulo 5.

La Figura 4.3 muestra de manera integrada el flujo general de interacción con la plataforma web según el rol del usuario. En este diagrama se visualizan las rutas disponibles para administradores, docentes y estudiantes desde el inicio de sesión, así como las funciones principales asociadas a cada perfil. Esta representación permite contextualizar las subsecciones siguientes, donde se detalla cada funcionalidad específica.

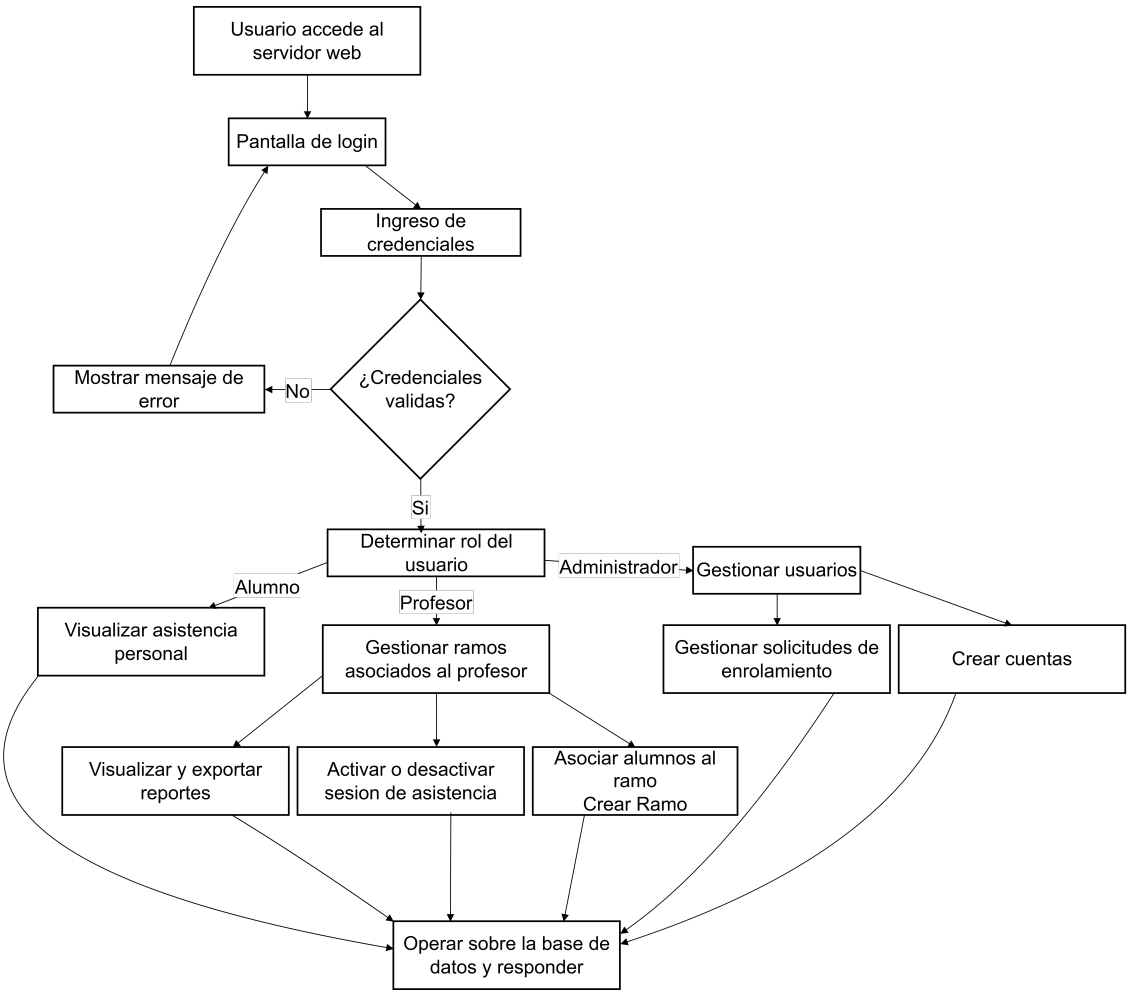


Figura 4.3: Flujo de interacción con la plataforma web según rol de usuario.

En el sistema coexisten tres perfiles principales: el administrador, el docente y el estudiante. El administrador es responsable de la creación y gestión de las cuentas de usuario, del mantenimiento general de la plataforma y del proceso de enrolamiento biométrico, dado que este procedimiento requiere control centralizado para asegurar la integridad y trazabilidad de la base de datos de

huellas. El docente administra sus asignaturas y consulta la asistencia registrada por el sistema, mientras que los estudiantes pueden acceder a su propia información de asistencia.

El diseño de esta interfaz se planteó con el objetivo de garantizar la autonomía y funcionalidad del sistema, evitando desde un inicio cualquier dependencia de integraciones externas con bases de datos institucionales. Considerar una conexión directa con los sistemas de gestión académica de la universidad habría implicado una serie de desafíos administrativos y técnicos —como la necesidad de autorizaciones formales, tiempos de espera inciertos y posibles restricciones de acceso— que no resultaban compatibles con los objetivos y el alcance definidos para este proyecto.

Por esta razón, se optó por una solución autogestionada, en la cual el sistema otorga a cada docente la facultad de crear sus propios asignaturas y asociar a los alumnos correspondientes. Esta decisión permitió implementar una herramienta simple, funcional y fácilmente extensible, sin depender de factores externos.

4.4.1. Acceso y autenticación

El acceso al sistema comienza en el perfil de administrador, quien dispone de una cuenta creada al momento de la instalación y es el encargado de generar las credenciales iniciales para docentes y estudiantes. Una vez creadas estas cuentas, cada usuario accede a su panel correspondiente mediante autenticación individual.

Los docentes ingresan utilizando credenciales únicas compuestas por su correo electrónico y una contraseña previamente registrada. Tras la autenticación, el sistema restringe su visualización exclusivamente a los cursos asociados a su cuenta y a los registros de asistencia correspondientes, garantizando que cada docente acceda únicamente a su información autorizada.

En el caso de los estudiantes, sus cuentas son creadas exclusivamente por el administrador. Ellos no crean ni modifican su propia información, y su acceso se limita a la visualización de sus registros personales de asistencia.

La validación de credenciales se realiza contra la base de datos interna de *BioAsist*, y la autenticación se gestiona mediante sesiones PHP. Estas sesiones aseguran que, una vez iniciada la sesión, el usuario mantenga acceso solo a la información que le corresponde según su rol. Además, el sistema incluye la funcionalidad para cerrar sesión y finalizar de manera segura el acceso del usuario.

La Figura 4.4 muestra el formulario de creación de cuentas desde la vista del administrador, mientras que la Figura 4.5 ilustra el panel principal del docente al iniciar sesión en el sistema.

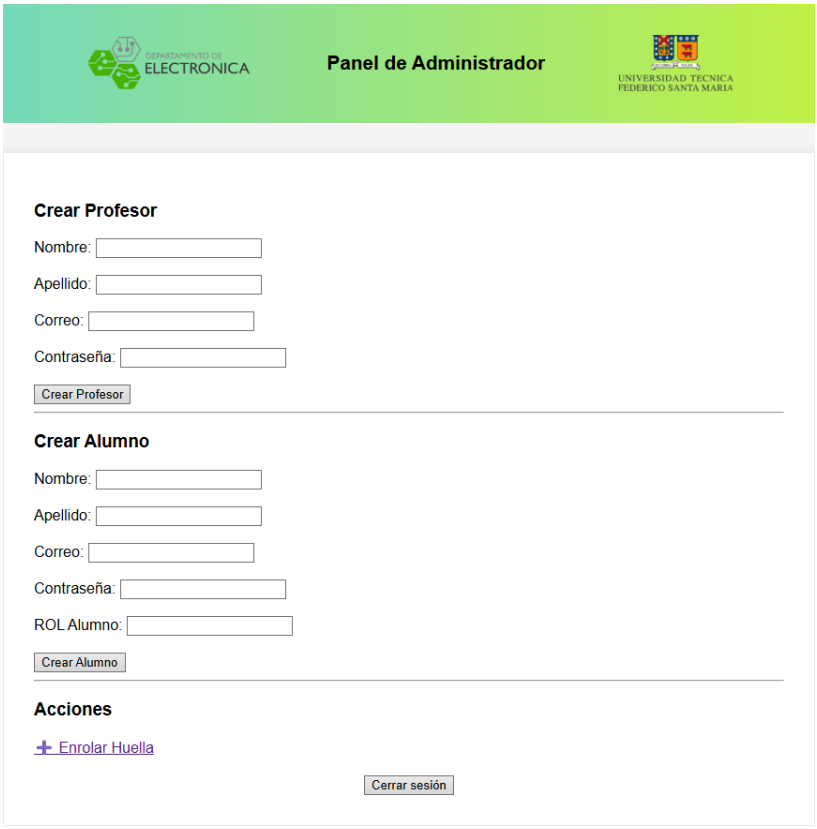


Figura 4.4: Formulario de creación de cuentas desde la vista del administrador

Figura 4.5: Panel principal del docente

4.4.2. Enrolamiento de huellas

El proceso de enrolamiento biométrico es una responsabilidad exclusiva del administrador, dado que constituye un procedimiento formal de registro que requiere control centralizado para asegurar la integridad y trazabilidad de la base de datos de huellas. Desde su panel, el administrador puede seleccionar al usuario que debe ser enrolado —generalmente un estudiante o un docente— y activar el proceso correspondiente en el sistema.

Una vez habilitado el enrolamiento, la plataforma registra la solicitud en la base de datos interna, asociándola al *id* del usuario, la fecha y hora, y un estado inicial de *pendiente*. El dispositivo BioAsist-Device consulta periódicamente este estado y, al detectar una solicitud activa, inicia la secuencia de captura biométrica utilizando el sensor óptico R307. Durante este procedimiento, el dispositivo guía al usuario mediante mensajes locales en pantalla, mientras que el sistema web muestra el estado general del proceso al administrador.

Si el enrolamiento se completa exitosamente, el dispositivo genera un `finger_id` —un identificador numérico asignado internamente por el sensor y utilizado únicamente para futuras verificaciones— y lo envía al servidor junto con la confirmación de éxito. La plataforma actualiza entonces el registro del usuario, marcando la solicitud como *completada* y almacenando el `finger_id` correspondiente.

El administrador también puede cancelar el proceso en caso de haber seleccionado al usuario incorrecto o ante cualquier situación operativa que lo requiera. En dicha situación, la solicitud se marca como *cancelada*, y el BioAsist-Device interrumpe inmediatamente cualquier acción en curso, quedando en estado de espera para próximas solicitudes.

Este flujo contempla de manera implícita un conjunto de estados —pendiente, procesando, completado y cancelado— que permiten estructurar de forma clara la interacción entre el dispositivo y el servidor. Si bien no se detallan aquí en forma de diagrama, estos estados definen el comportamiento esperado del sistema ante cada transición, asegurando consistencia durante el proceso de enrolamiento.

La figura 4.6 muestra la interfaz de enrolamiento de huellas disponible en el panel del administrador.

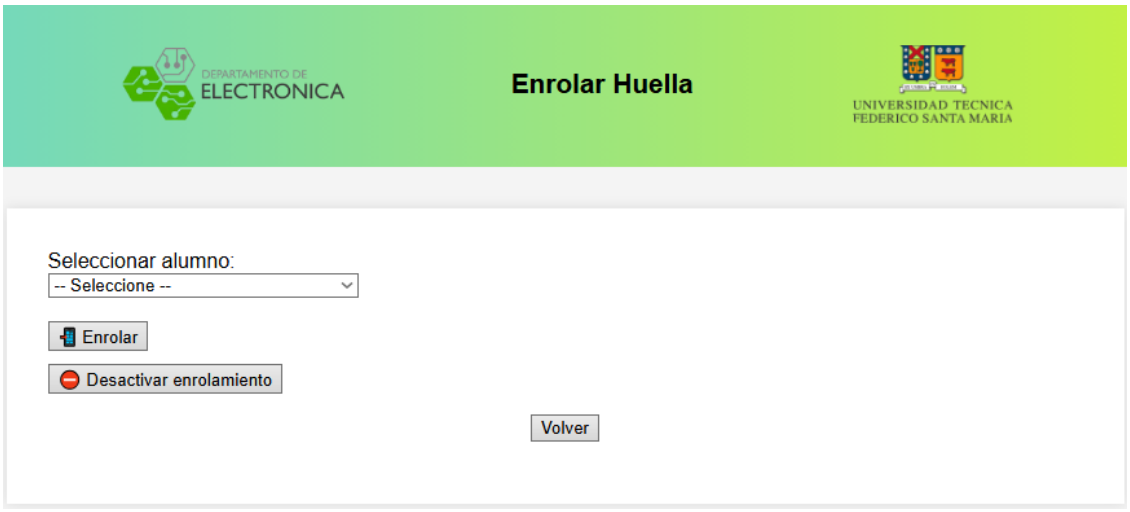


Figura 4.6: Interfaz de enrolamiento de huellas desde el panel del administrador

4.4.3. Funciones del Docente

Crear nueva asignatura

Esta funcionalidad permite al docente registrar un nuevo ramo dentro de *BioAsist*. Cada ramo representa una instancia específica de una asignatura, la cual puede incluir uno o más paralelos. Para reflejar esta estructura académica, el sistema permite registrar de manera explícita el número de paralelo asociado al ramo, lo que facilita la organización independiente de cada sección y el control diferenciado de asistencia para cada grupo de estudiantes.

Al crear un ramo, el docente debe ingresar su nombre, el código identificador de la asignatura (por ejemplo, EL0308) y el número de paralelo correspondiente. Una vez registrada esta información, el ramo queda automáticamente vinculado al docente que realizó la creación, apareciendo de inmediato en su panel tanto para el registro de asistencia como para su posterior visualización.

Desde el punto de vista de la base de datos, cada ramo se almacena con cuatro atributos principales: el nombre del ramo, su código, el número de paralelo y el identificador del profesor responsable. Este último se implementa mediante una clave foránea hacia la tabla `usuarios`, lo que asegura la integridad referencial y garantiza que cada ramo esté asociado a un docente válido.

Es importante destacar que, en esta etapa, el sistema aún no conoce qué estudiantes pertenecen a cada ramo. Esta asociación se realiza posteriormente mediante la funcionalidad *Inscribir alumnos a ramo*, donde el docente selecciona explícitamente a los alumnos que participarán en la asignatura o paralelo correspondiente.

La Figura 4.7 muestra el formulario de creación de asignaturas y la vinculación automática al docente que los registra.

Figura 4.7: Formulario de creación de asignaturas y vinculación automática al docente

Registrar asistencia

La funcionalidad de registrar asistencia permite al docente iniciar y finalizar el registro de asistencia de sus estudiantes de manera remota. El proceso se basa en un mecanismo de cambio de estado: cuando el profesor habilita la toma de asistencia para un ramo determinado, la plataforma actualiza un indicador en la base de datos que es consultado periódicamente por el *BioAsist-Device*. Al detectar que la asistencia ha sido activada, el dispositivo cambia a su modo operativo de reconocimiento de huellas dactilares.

Para comenzar la sesión, el docente selecciona el ramo y paralelo mediante un menú desplegable que muestra únicamente los cursos asociados a su cuenta. Al presionar el botón de inicio, el sistema registra el comienzo de la sesión y habilita el dispositivo para recibir huellas. Conceptualmente, todos los estudiantes inscritos en el ramo se consideran inicialmente como ausentes, y su estado cambia a presente únicamente si su huella es reconocida correctamente durante el período activo de la sesión. En la implementación, al finalizar la sesión de asistencia se genera o actualiza un registro en la tabla `asistencia` para cada alumno inscrito, marcando como *Presente* a quienes fueron reconocidos al menos una vez por el dispositivo y como *Ausente* a quienes no registraron presencia.

Durante la sesión, la interfaz web muestra en tiempo real los registros de los alumnos que se han identificado mediante el lector biométrico, desplegando su nombre, rol, fecha y estado del alumno. Esta información permite al docente monitorear el avance del proceso de asistencia y verificar que los estudiantes se presenten efectivamente.

Una vez finalizada la toma de asistencia, el docente presiona el botón de detención. El sistema actualiza el estado de la sesión y el *BioAsist-Device* vuelve a su modo de espera. Para efectos del registro, se consideran presentes únicamente los estudiantes que hayan sido reconocidos por el dispositivo durante la ventana activa de asistencia; todos los demás permanecen como ausentes.

Esta dinámica puede representarse naturalmente como una máquina de estados, donde el sistema transita entre *inactivo*, *asistencia activa*, *procesando registros* y *asistencia finalizada*, con transiciones provocadas por acciones explícitas del docente y eventos de reconocimiento generados por el dispositivo. Esta estructura contribuye a una implementación más clara y robusta del proceso de asistencia.

La Figura 4.8 muestra la vista del panel de registro de asistencia con actualización en tiempo real.

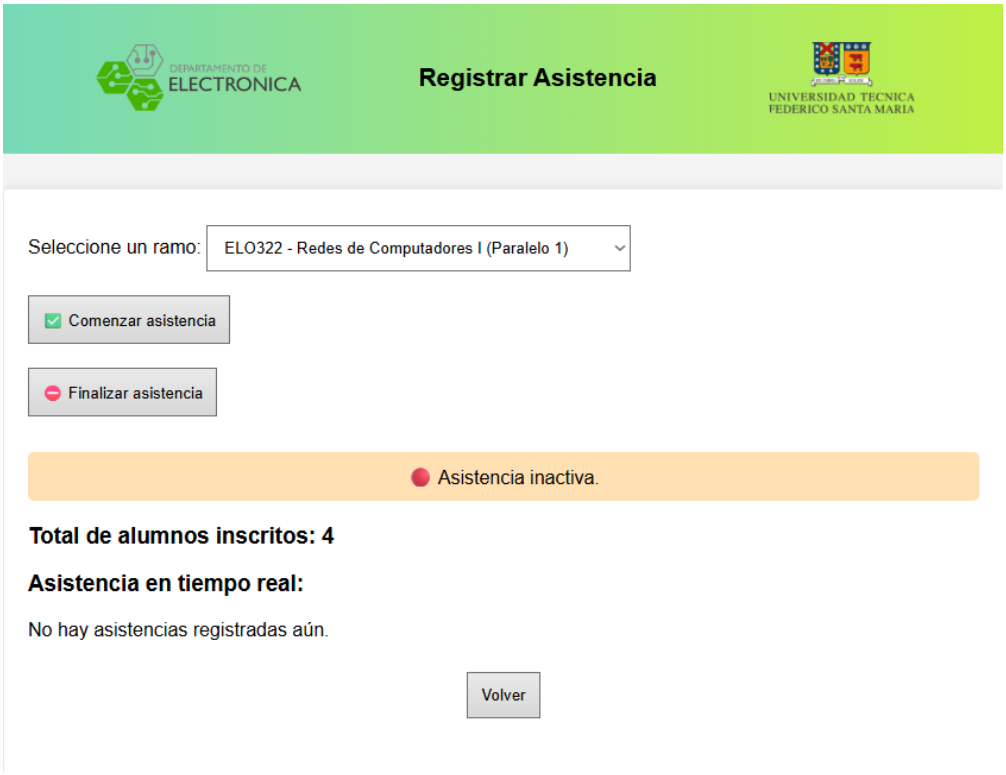


Figura 4.8: Vista del panel de registro de asistencia con actualización en tiempo real

Inscribir alumnos a ramo

Esta funcionalidad permite al docente inscribir uno o más alumnos en los cursos que tiene asignados. Esta inscripción es fundamental para que los estudiantes sean considerados por el sistema al momento de registrar asistencia, ya que la toma de asistencia se realiza únicamente sobre los alumnos asociados a cada ramo.

La interfaz presenta un listado completo de los estudiantes registrados en *BioAsist*. Para facilitar la búsqueda en cursos con grandes cantidades de alumnos, se incluye un campo de filtro por nombre, lo que permite al docente localizar rápidamente a los estudiantes que desea inscribir. Además, la selección es múltiple, permitiendo vincular a varios alumnos en una sola operación y así agilizar el proceso.

Una vez confirmada la selección, el sistema actualiza la tabla intermedia `alumnos_ramos`, que modela la relación muchos-a-muchos entre las entidades `usuarios` y `ramos`. Esta estructura mantiene la integridad referencial de la base de datos y asegura que cada alumno quede asociado únicamente a los ramos en los que participa.

La Figura 4.9 muestra la interfaz utilizada para inscribir alumnos en los ramos dictados por el docente.

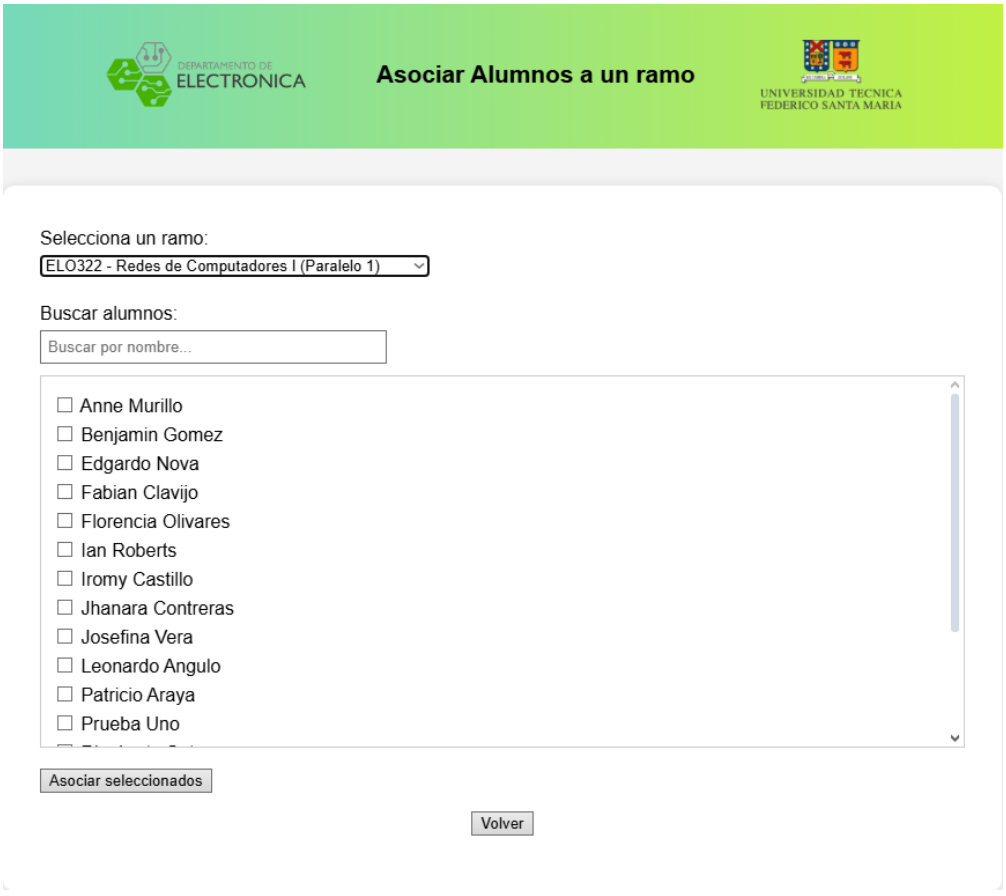


Figura 4.9: Interfaz para inscribir alumnos en ramos dictados por el docente

Gestionar ramos

Esta funcionalidad permite al docente mantener y corregir la configuración de los ramos que ha creado, otorgándole un mayor control sobre su información académica y facilitando la rectificación de errores ingresados durante la etapa inicial de configuración.

Desde esta vista, el docente puede modificar el nombre del ramo o actualizar su número de paralelo, en caso de haber registrado uno incorrectamente. Asimismo, es posible visualizar el listado de estudiantes actualmente inscritos en cada ramo y, si es necesario, desvincular a alguno de ellos, por ejemplo, si fue asociado por error.

Adicionalmente, se incluye la opción de eliminar un ramo, con la restricción de que esta acción sólo puede realizarse si el ramo no posee registros de asistencia asociados. Esta medida garantiza la coherencia y trazabilidad de los datos históricos de *BioAsist*, evitando la pérdida de información académica pasada.

La Figura 4.10 muestra la interfaz de gestión que permite modificar o eliminar ramos, así como revisar o ajustar las inscripciones de estudiantes vinculados a cada curso.

Figura 4.10: Interfaz de gestión para modificar o eliminar ramos y administrar inscripciones

Visualización de la asistencia

La interfaz web de *BioAsist* permite al docente acceder a los registros de asistencia correspondientes a los ramos y paralelos que tiene asignados. Durante la toma de asistencia en tiempo real, el sistema despliega un listado dinámico de los estudiantes que han registrado su presencia mediante el lector biométrico, permitiendo al docente monitorear el proceso mientras está en curso.

Además, el sistema ofrece una sección específica para la visualización histórica de asistencia por ramo. A través del apartado *Ramos que dicta*, el docente puede consultar la asistencia acumulada de sus estudiantes y aplicar distintos filtros por rango de fechas (día, semana o mes). Esta funcionalidad facilita el seguimiento del desempeño académico y la detección de patrones de inasistencia.

El registro mostrado distingue entre estudiantes presentes y ausentes. Un alumno es clasificado como presente si su huella fue reconocida durante la sesión correspondiente; en caso contrario, se considera ausente, de acuerdo con la lógica de funcionamiento descrita en la sección de toma de asistencia.

Finalmente, se incluye la opción de descargar los registros en formato Excel (`.csv`), permitiendo respaldos locales, análisis complementarios por parte del docente o su posterior integración con plataformas externas como Moodle. La Figura 4.11 muestra un ejemplo de esta visualización para un ramo específico.

Figura 4.11: Visualización de la asistencia por ramo

4.5. Diseño de la base de datos

La base de datos desarrollada para *BioAsist* cumple la función de almacenar, gestionar y organizar la información relacionada con los usuarios (profesores y alumnos), las asignaturas, los respectivos registros de asistencia, los enrolamientos biométricos y los controles de toma de asistencia. Este diseño busca garantizar la integridad de los datos, la trazabilidad de las acciones realizadas y una estructura escalable para futuras mejoras o aplicaciones.

4.5.1. Modelo entidad-relación

La base de datos se diseñó utilizando el enfoque entidad-relación, normalizado hasta Tercera Forma Normal (3FN), lo cual permite una estructura coherente, libre de redundancias y con relaciones bien definidas entre entidades. En la Figura 4.12 se muestra el diagrama del modelo entidad-relación implementado.

4.5.2. Descripción de tablas principales

- **usuarios**: Esta tabla centraliza a todos los actores del sistema (estudiantes, docentes y administrador). Se distingue el perfil de cada usuario mediante el campo `tipo`, que indica su rol dentro de *BioAsist*. Además de este campo, la tabla almacena nombre, apellido, correo electrónico, contraseña y, en el caso de los estudiantes, su identificador académico (`rol_alumno`) y el `finger_id` asociado a la huella enrolada.

  En lugar de mantener tablas separadas para alumnos y profesores, se optó por un diseño unificado basado en la tabla `usuarios`, lo que evita la duplicación de campos comunes y simplifica la lógica de autenticación y gestión de credenciales. Para la clave primaria se utiliza un identificador entero auto–incremental (`id`), que actúa como identificador técnico interno. Este diseño permite, por ejemplo, manejar usuarios sin RUT asociado (como el administrador) y reducir el acoplamiento entre la clave lógica del usuario y las referencias internas en la base de datos. En el caso de los estudiantes, el `rol_alumno` se maneja como un identificador institucional adicional, sobre el cual pueden definirse restricciones de unicidad si se requiere.

- **ramos**: Contiene la información de cada asignatura impartida por el sistema, con campos como nombre, código (por ejemplo, EL0308), número de paralelo y el identificador del profesor responsable (`id_profesor`), que actúa como clave foránea hacia la tabla `usuarios`. Cada registro de esta tabla representa una combinación específica de asignatura y paralelo dictada por un único docente, lo que facilita la gestión independiente de asistencia e inscripciones para cada sección. En un despliegue institucional, el modelo podría extenderse para permitir múltiples docentes asociados a un mismo ramo mediante una tabla intermedia adicional, pero para el prototipo desarrollado se considera un profesor principal por cada ramo.

- **alumnos_ramos**: Esta tabla intermedia modela la relación muchos a muchos entre alumnos y ramos. Un alumno puede estar inscrito en múltiples ramos y un ramo puede tener múltiples alumnos. Al separar esta relación en dos relaciones uno-a-muchos, se asegura la conformidad con la 1FN y se evita almacenar múltiples valores en un solo campo.

- **asistencia**: Registra la asistencia de un alumno a un ramo en una fecha específica. Cada registro almacena el identificador del alumno, el identificador del ramo, la fecha de la sesión y un campo `estado` que puede tomar valores como *Presente* o *Ausente*. Durante la toma de asistencia, el sistema va marcando como presentes a los estudiantes reconocidos por el dispositivo. Al finalizar la sesión, se completa o actualiza un registro para cada alumno inscrito en el ramo, de modo que aquellos que no fueron reconocidos durante el período activo queden registrados explícitamente como ausentes. Este diseño simplifica las consultas posteriores y la generación de reportes, al contar con un registro explícito del estado de cada estudiante en cada sesión.

- **estado_asistencia**: Permite habilitar o deshabilitar la toma de asistencia en un ramo determinado. Cada registro asocia un ramo con una sesión de asistencia, e incluye un campo `activo` que indica si la sesión se encuentra en curso, junto con las marcas de tiempo de inicio y término. Cuando el docente inicia la toma de asistencia desde la interfaz web, se crea o actualiza el registro correspondiente marcándolo como activo; al finalizar la sesión, este mismo registro se actualiza para indicar su término. El *BioAsist-Device* consulta periódicamente esta tabla para determinar si debe operar en modo de registro de asistencia o permanecer en modo de espera.
- **enrolamientos**: Almacena los eventos asociados al proceso de registro biométrico. Cada registro incluye el identificador del usuario que debe ser enrolado, la fecha y hora de la solicitud, y un campo de `estado` que puede tomar los valores *pendiente*, *procesando*, *completado* o *cancelado*. Esta tabla es consultada periódicamente por el BioAsist-Device, el cual inicia o detiene el proceso de enrolamiento según la transición del estado registrada en el servidor.

  Es importante señalar que el sistema no almacena ni transmite datos biométricos sensibles. El sensor R307 se encarga internamente de la extracción y procesamiento de la huella, generando únicamente un identificador numérico (`finger_id`) que es válido sólo dentro del propio módulo. Este valor no permite reconstruir la huella original y, por lo tanto, no constituye información biométrica en sí misma. De esta forma, el servidor sólo recibe el `finger_id` final una vez completado el proceso, lo que simplifica la gestión de seguridad y evita la necesidad de manejar plantillas biométricas o aplicar técnicas avanzadas de cifrado en la base de datos.

La Figura 4.12 presenta el modelo entidad-relación de la base de datos desarrollado para *BioAsist*.

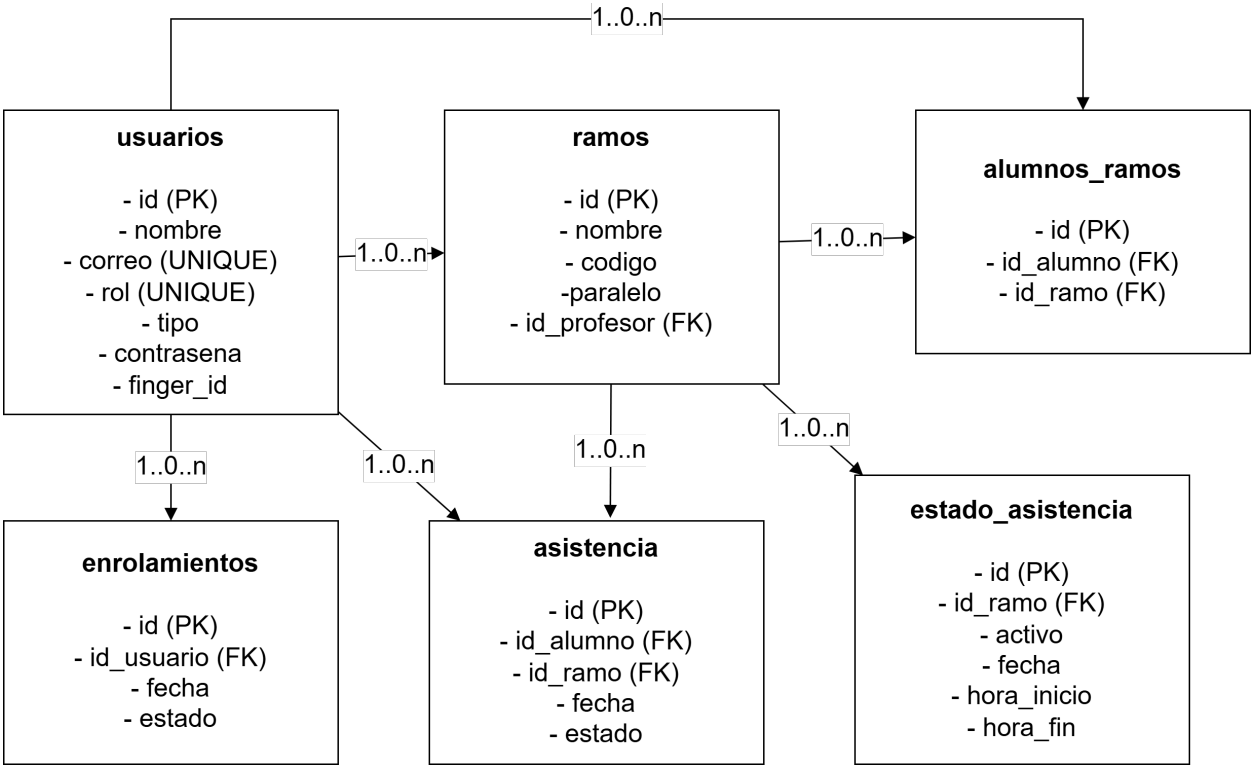


Figura 4.12: Diagrama del modelo entidad-relación de la base de datos

El modelo de datos presentado en esta sección corresponde al diseño utilizado en el prototipo actual de *BioAsist*, implementado para operar con un único dispositivo biométrico y un conjunto acotado de cursos. No obstante, en un escenario de despliegue real a nivel institucional podrían coexistir múltiples *BioAsist-Device* distribuidos en distintas salas o edificios, lo que introduce nuevos requerimientos de escalabilidad y control.

Para soportar este tipo de escenarios, el modelo debería incorporar información adicional que permita identificar el origen de cada solicitud y administrar múltiples sesiones de asistencia en paralelo. Entre las extensiones posibles se encuentran:

- **Incorporar una tabla de dispositivos** que permita registrar cada BioAsist-Device mediante un identificador único, su ubicación física y su estado de operación. Esto facilitaría auditorías, monitoreo y detección de fallos.
- **Asociar las sesiones de asistencia a un dispositivo específico**, agregando un campo `id_device` en la tabla `estado_asistencia`. Esto evitaría colisiones cuando distintas salas abran sesiones simultáneas para cursos distintos.
- **Controlar el enrolamiento desde un punto centralizado**, registrando qué dispositivo ejecuta cada proceso y evitando conflictos en la asignación de `finger_id`.
- **Permitir múltiples ramos con códigos repetidos en distintos dispositivos**, utilizando el campo `paralelo` para diferenciar cada grupo.

Estas consideraciones se relacionan directamente con los escenarios de escalamiento discutidos en la Sección 4.3, donde se analiza el impacto de operar con múltiples dispositivos y procesos concurrentes de asistencia y enrolamiento.

4.5.3. Relaciones entre tablas

Las relaciones entre las entidades se muestran en la Tabla 4.1. Estas relaciones fueron implementadas utilizando claves foráneas para asegurar la integridad referencial en todas las operaciones.

Tabla 4.1: Relaciones entre tablas de la base de datos

Tabla origen	Campo	Tabla destino	Tipo de relación
ramos	id_profesor	usuarios	1:N (profesor a ramos)
alumnos_ramos	id_alumno	usuarios	N:1
alumnos_ramos	id_ramo	ramos	N:1
asistencia	id_alumno	usuarios	N:1
asistencia	id_ramo	ramos	N:1
estado_asistencia	id_ramo	ramos	N:1
enrolamientos	id_usuario	usuarios	N:1

4.5.4. Decisiones de diseño y normalización

Durante el diseño del modelo se aplicaron las tres primeras formas normales:

- **1FN:** Se evitó el uso de atributos multivaluados. Todos los campos contienen valores atómicos.

- **2FN:** Las tablas no tienen dependencias parciales respecto de claves primarias compuestas. Los atributos dependen de la totalidad de la clave.
- **3FN**: No se almacenan atributos que dependan transitivamente de una clave primaria. Por ejemplo, el identificador biométrico `finger_id` se almacena únicamente en la tabla `usuarios` y no se replica en otras tablas que hacen referencia al alumno.

  En el contexto de la información biométrica, se evaluó la posibilidad de almacenar plantillas de huellas en la base de datos. Aunque el sensor óptico R307 permite extraer la plantilla en forma de un arreglo binario, esto implicaría manejar datos altamente sensibles que requerirían cifrado, almacenamiento seguro y mecanismos de comparación externos, aumentando significativamente la complejidad del sistema. Por esta razón, el modelo de datos se limita a registrar el identificador numérico `finger_id` generado por el propio sensor, el cual se utiliza como referencia interna para las verificaciones sin necesidad de persistir plantillas biométricas en la base de datos.

Capítulo 5

Implementación

Este capítulo presenta la implementación del sistema web asociado a *BioAsist*, abarcando el desarrollo de la interfaz gráfica y la lógica del backend que sustenta su funcionamiento. A diferencia del capítulo anterior, centrado en el diseño conceptual, aquí se describe cómo dichas decisiones fueron materializadas en código, detallando la estructura de las vistas, la gestión de usuarios y ramos, y los mecanismos que permiten el enrolamiento y registro de asistencia. También se expone la API REST que comunica al servidor con el BioAsist-Device, junto con los endpoints y flujos que coordinan esta interacción. En conjunto, el capítulo documenta la arquitectura final del sistema web implementado.

5.1. Descripción del hardware utilizado

BioAsist fue diseñado a partir de elementos de bajo costo y fácil disponibilidad, los cuales en conjunto permiten enrolar y procesar huellas dactilares, comunicarse con un servidor web y ofrecer retroalimentación al usuario mediante el *BioAsist-Device*.

5.1.1. Microcontrolador ESP32

El **ESP32 DevKit V1** corresponde a una placa de desarrollo basada en el microcontrolador ESP32, diseñada por *Espressif Systems* [3]. Este dispositivo fue seleccionado para el proyecto debido a que combina un tamaño reducido con conectividad inalámbrica integrada y un ecosistema de desarrollo ampliamente documentado. A diferencia de otras opciones como el ESP8266 o microcontroladores STM32 con módulos Wi-Fi externos, la ESP32 ofrece una solución más completa y con menor complejidad de integración dentro del *BioAsist-Device*.

Entre las razones de su elección destacan:

- **Conectividad inalámbrica:** Incluye Wi-Fi 802.11 b/g/n y Bluetooth 4.2 (con soporte BLE), lo cual permite la comunicación directa con el servidor web de *BioAsist* sin necesidad de hardware adicional.
- **Versatilidad en periféricos:** Cuenta con múltiples interfaces de comunicación (UART, I2C, SPI, ADC, DAC, PWM), que facilitan la conexión a diversos sensores y dispositivos. En este proyecto se empleó UART2 para el lector R307 y el bus I²C para la pantalla LCD.
- **Capacidad de procesamiento:** Su arquitectura de doble núcleo permite ejecutar de forma paralela las tareas de comunicación y control del sensor biométrico, garantizando tiempos de respuesta adecuados.

- **Soporte de software:** Dispone de un extenso ecosistema de librerías en entornos como Arduino IDE [13], PlatformIO [14] y ESP-IDF [15], lo que agiliza el desarrollo y reduce el riesgo de problemas de compatibilidad.
- **Disponibilidad y costo:** Es un módulo ampliamente distribuido y de bajo costo, lo que lo convierte en una alternativa accesible para proyectos académicos y prototipos funcionales.

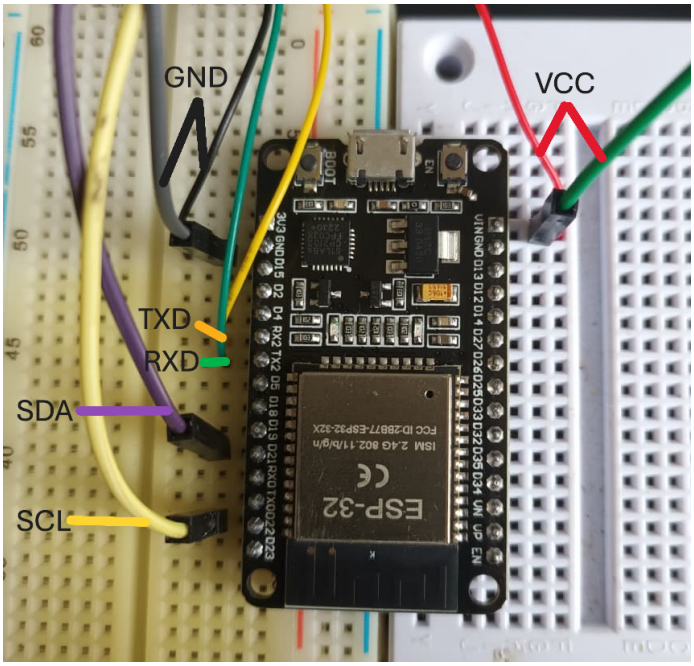
En la Tabla 5.1 se presentan las características técnicas más relevantes del ESP32 DevKit V1, seleccionadas en función de su impacto en *BioAsist*.

Tabla 5.1: Características técnicas principales del ESP32 DevKit V1

Parámetro	**Valor típico**
Procesador	Tensilica Xtensa LX6 dual-core, 240 MHz
Memoria SRAM interna	520 KB
Memoria Flash externa	4 MB (dependiente de la placa)
Conectividad inalámbrica	Wi-Fi 802.11 b/g/n, Bluetooth v4.2 (BLE)
Interfaces de comunicación	UART, I2C, SPI, ADC, DAC, PWM
Voltaje de operación	3.0 – 3.6 V
Consumo típico (Wi-Fi activo)	160 mA aprox.

La Figura 5.1a muestra la placa ESP32 DevKit V1 utilizada en el prototipo, mientras que en la Figura 5.1b se presenta un diagrama de sus conexiones hacia los periféricos principales empleados en este proyecto (sensor biométrico y pantalla LCD), que se describen con mayor detalle en las subsecciones siguientes.

(a) Módulo ESP32 DevKit V1.



(b) Conexiones principales de la ESP32 hacia los periféricos del dispositivo.

Figura 5.1: ESP32 DevKit V1 y sus conexiones hacia los periféricos del *BioAsist-Device*.

5.1.2. Lector de huella R307

El sensor biométrico **R307** corresponde a un módulo óptico de captura y reconocimiento de huellas dactilares, ampliamente utilizado en aplicaciones de control de acceso y sistemas embebidos debido a su bajo costo y facilidad de integración. Este dispositivo realiza tanto la adquisición de imágenes como el procesamiento para la generación de plantillas biométricas, permitiendo un uso autónomo sin necesidad de hardware adicional especializado.

Las funciones principales del módulo pueden resumirse en los siguientes puntos:

- **Enrolamiento de huella:** Para este proceso el usuario debe ingresar su huella dos veces. El *BioAsist-Device* procesa ambas imágenes, genera una plantilla basada en el análisis de características y la almacena en memoria interna del lector, asignándole un identificador único para facilitar su gestión posterior.
- **Verificación 1:1:** Consiste en comparar una huella recién capturada contra una plantilla específica almacenada en el módulo. En este modo, la verificación es realizada internamente por el propio lector, utilizando la plantilla indicada. Este mecanismo forma parte de las capacidades nativas del R307 y opera de manera autónoma respecto del sistema *BioAsist*.
- **Identificación 1:N:** La huella capturada se compara contra todas las plantillas almacenadas en la memoria del módulo hasta encontrar una coincidencia. Este es el método empleado en *BioAsist*, ya que permite identificar directamente al estudiante sin requerir que ingrese previamente un identificador.
- **Interfaz de comunicación:** El módulo soporta comunicación serial mediante UART a niveles lógicos TTL, con una velocidad de transferencia configurable (por defecto, `57600 bps`). Alternativamente, algunas versiones del R307 incluyen soporte para conexión vía USB 2.0. En este proyecto se empleó la interfaz UART, utilizando los pines de transmisión (TXD) y recepción (RXD) del módulo.

En la Tabla 5.2 se resumen las especificaciones técnicas más relevantes del R307.

Tabla 5.2: Características técnicas principales del sensor R307

Parámetro	**Valor típico**
Método de captura	Óptico (500 dpi)
Capacidad de almacenamiento	1000 plantillas
Tamaño de plantilla	512 bytes
Tiempo de adquisición	0.5 s
Tiempo de búsqueda (1:N)	Hasta 1 s
Interfaz de comunicación	UART TTL (3.3/5 V), USB 2.0 opcional
Velocidad por defecto	57600 bps
Voltaje de operación	4.2–6 V DC
Consumo típico	50 mA
Dimensiones	54 mm × 20 mm × 20 mm

Pines y conexiones hacia la ESP32

En la Figura 5.2 se presenta el diagrama de pines externos del R307, extraído de su datasheet, donde se identifican las señales principales utilizadas en la integración con el *BioAsist-Device*.

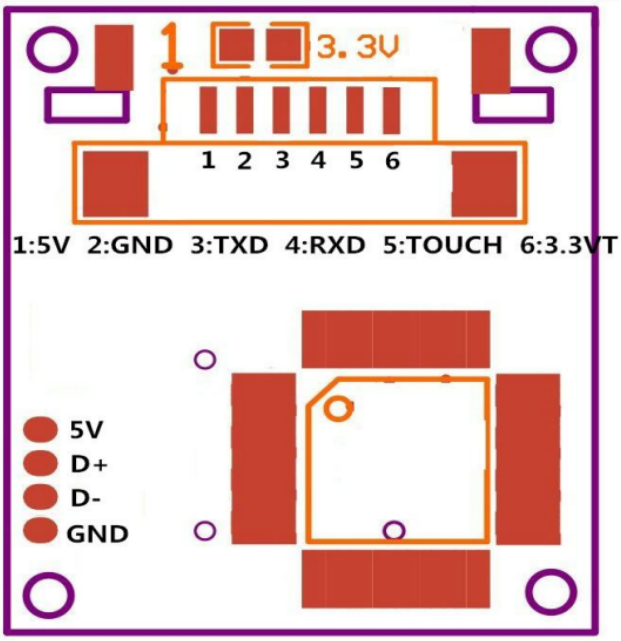


Figura 5.2: Diagrama de pines externos del módulo R307 (adaptado de [2]).

Para la conexión con el ESP32 se emplean únicamente los cuatro primeros pines del módulo: alimentación a 5V, tierra común (GND), transmisión (TXD) y recepción (RXD). El pin 5, denominado TOUCH, permite implementar mecanismos de ahorro energético, ya que se activa únicamente cuando el sensor detecta contacto con un dedo, pudiendo despertar al microcontrolador o iniciar una rutina sin necesidad de mantener el sistema activo de manera permanente.

Aunque esta funcionalidad ofrece ventajas en términos de eficiencia energética, no fue utilizada en este proyecto, principalmente por razones de simplicidad y alcances, además de que los ejemplos de referencia empleados durante el desarrollo no incorporaban esta característica.

En la Figura 5.3 se ilustran las conexiones del R307 hacia el ESP32.

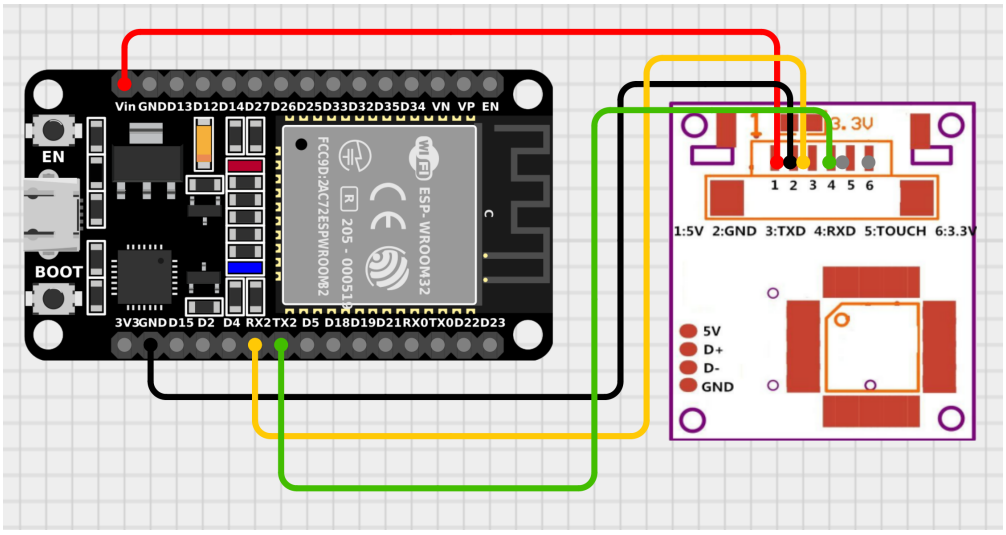
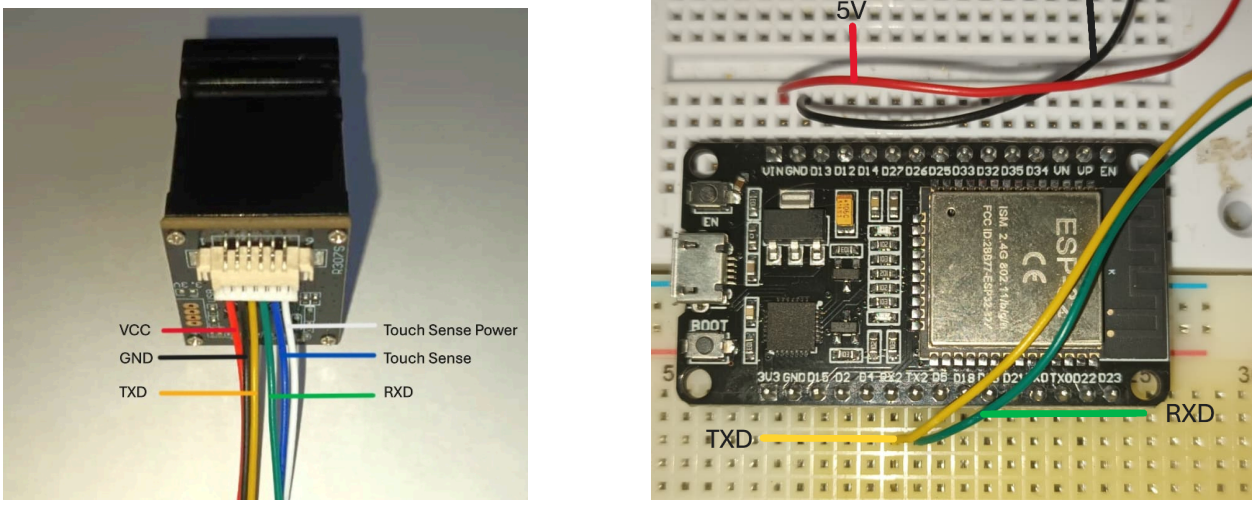


Figura 5.3: Diagrama de conexión entre el R307 y la ESP32.

La implementación física del montaje puede observarse en la Figura 5.4, donde se muestra tanto el módulo biométrico como su conexión a la ESP32 dentro del *BioAsist-Device*.



(a) Módulo biométrico R307. (b) Conexión del R307 con la ESP32.

Figura 5.4: Lector de huellas R307 y su conexión física al microcontrolador ESP32.

Cabe señalar que el R307 dispone de un conjunto de comandos internos para enrolar, buscar, eliminar plantillas y consultar parámetros de configuración. En este proyecto se empleó la librería *Adafruit Fingerprint* [12], que abstrae dichos comandos y facilita la integración con la ESP32. Los detalles de esta implementación se presentan en la sección 5.2.

Finalmente, la elección del R307 se basó en un análisis comparativo frente a otros sensores disponibles, tales como el AS608, GT-521F32 y los módulos Adafruit ID 751 e ID 4651 [16–19]. Si bien algunos de estos dispositivos ofrecen ventajas específicas —como tiempos de captura más rápidos (Adafruit ID 4651), menor consumo energético (AS608) o mayor capacidad de almacenamiento (GT-521F32 con hasta 3000 plantillas)—, también presentan desventajas como menor disponibilidad local, mayor costo o una integración menos directa con microcontroladores de bajo costo. En contraste, el R307 ofreció un equilibrio adecuado entre capacidad de almacenamiento (1000 plantillas), tiempo de respuesta, documentación técnica y compatibilidad con la librería *Adafruit Fingerprint*, lo que facilitó su integración con la ESP32 y redujo el tiempo de desarrollo de *BioAsist*.

5.1.3. Pantalla LCD 16x2 con interfaz I2C

El *BioAsist-Device* incorpora una **pantalla LCD 16x2**, cuya función principal es proveer retroalimentación visual al usuario durante el enrolamiento y verificación de huellas. Este tipo de display, basado en el controlador estándar *Hitachi HD44780* [20], permite mostrar hasta dos líneas de 16 caracteres alfanuméricos cada una.

Con el objetivo de simplificar el cableado y reducir la cantidad de pines utilizados en el microcontrolador, se empleó una versión con **interfaz I2C**, la cual incorpora un expansor de pines *PCF8574* [21]. Este módulo permite controlar la pantalla mediante únicamente dos señales (SDA y SCL), en contraste con las seis a ocho líneas de datos requeridas por la versión paralela. Gracias a ello, se optimiza el uso de pines GPIO de la ESP32 y se facilita el montaje físico del prototipo.

Las principales características del display se resumen en la Tabla 5.3.

Tabla 5.3: Características técnicas principales de la pantalla LCD 16x2 I2C

Parámetro	**Valor típico**
Tecnología	LCD basado en controlador HD44780
Capacidad de visualización	2 líneas × 16 caracteres
Interfaz de comunicación	I2C mediante PCF8574
Número de pines al microcontrolador	2 (SDA, SCL)
Voltaje de operación	5 V (compatible con lógica I2C a 3.3 V)
Consumo típico	1–2 mA (sin retroiluminación)
Dimensiones aproximadas	80 mm × 36 mm

La Figura 5.5 muestra el módulo LCD con interfaz I2C utilizado en el prototipo, junto con las anotaciones correspondientes a los cuatro pines principales del módulo: GND, VCC, SDA y SCL.

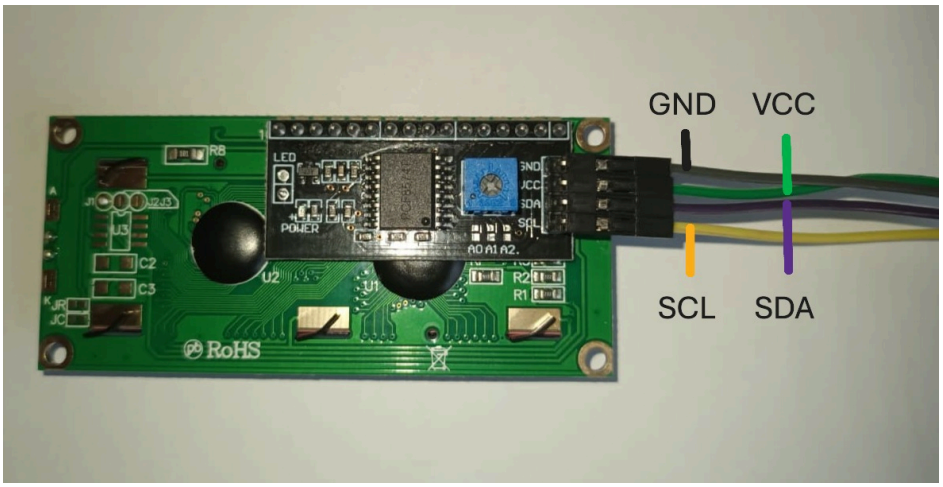


Figura 5.5: Pantalla LCD 16x2 con módulo I2C y conexiones utilizadas en *BioAsist*.

La conexión hacia la ESP32 se realiza de acuerdo con la siguiente asignación:

- **GND** → pin GND de la ESP32.
- **VCC** → pin VIN/5V para alimentar el módulo.
- **SDA** → GPIO21 (línea de datos I2C).
- **SCL** → GPIO22 (línea de reloj I2C).

En la Figura 5.6 se presenta el diagrama esquemático de la conexión entre la pantalla LCD y la ESP32, empleando únicamente dos líneas de comunicación además de alimentación y tierra.

Gracias al soporte ofrecido por librerías ampliamente usadas en entornos como Arduino/ESP32, tales como `LiquidCrystal_I2C` [22], la integración de este módulo resultó directa, permitiendo desplegar mensajes de estado como confirmaciones de enrolamiento, validación de asistencias y reportes de error de manera clara y eficiente.

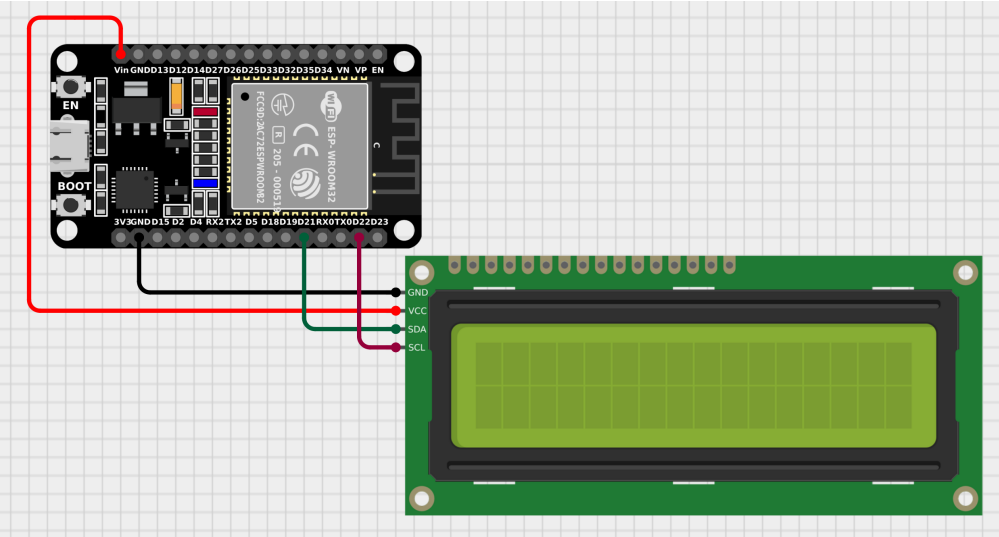


Figura 5.6: Diagrama de conexión entre la pantalla LCD 16x2 (I2C) y la ESP32.

5.1.4. Integración general del *BioAsist-Device*

Una vez descritos los periféricos individuales —el lector biométrico R307, la placa ESP32 y la pantalla LCD 16x2 con interfaz I2C— resulta necesario presentar la forma en que estos componentes se integran en el prototipo final. En la Figura 5.7 se muestra el **diagrama completo de conexiones** del *BioAsist-Device*, el cual resume en un único esquema todas las líneas de alimentación y señales utilizadas por el sistema. Este diagrama permite visualizar cómo se interconectan los módulos principales, facilitando tanto la replicación del montaje como la comprensión del flujo de datos entre los distintos elementos del hardware. Asimismo, la figura distingue claramente las interfaces empleadas: UART2 para la comunicación entre la ESP32 y el lector R307, I2C para el control de la pantalla LCD, y las líneas de alimentación provenientes de la fuente externa de 5 V.

5.1.5. Alimentación mediante powerbank

Para la alimentación del *BioAsist-Device* se optó por un powerbank comercial *Hoco® J115 Mini* de 5000 mAh, con salida USB de 5 V. Esta decisión se tomó por su disponibilidad, portabilidad y compatibilidad directa con los requerimientos del lector biométrico R307, el cual opera a 5 V.

El powerbank entrega una tensión estable de 5 V que permite energizar tanto el sensor de huellas como a la ESP32 (a través de su pin VIN/5V) y al display LCD 16x2, evitando la necesidad de fuentes adicionales. Con esta solución se simplifica la integración del prototipo, se facilita su transporte y se garantiza que el dispositivo pueda funcionar en escenarios sin acceso inmediato a una toma de corriente.

En caso de existir disponibilidad de 220 V, esta solución no permite conectarse directamente a la red eléctrica, ya que el prototipo no incorpora un sistema de conversión ni protección para reducir la tensión de 220 V AC a los 5 V DC requeridos por la ESP32.

Cabe señalar que existen alternativas más elaboradas, como el diseño de un sistema de alimentación basado en baterías recargables tipo *DIY battery pack* con circuitos de carga y protección propios. Sin embargo, este enfoque excede los objetivos planteados en la memoria, por lo que se optó por una solución práctica y suficiente para la etapa de implementación del prototipo.

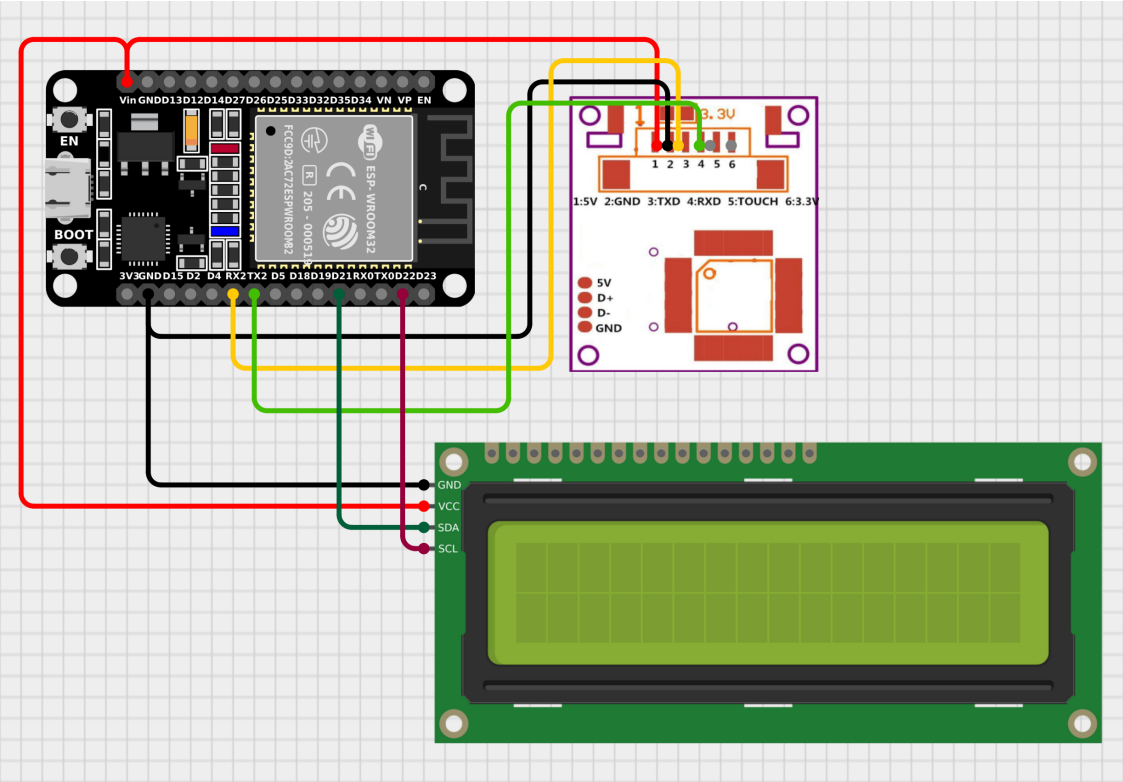


Figura 5.7: Diagrama general de conexiones del *BioAsist-Device*.

5.1.6. Costo de implementación del prototipo

Para complementar la descripción del hardware utilizado en el BioAsist-Device, a continuación se presenta un desglose del costo total de implementación del prototipo. Los valores corresponden a precios de mercado obtenidos durante el periodo de desarrollo, expresados en pesos chilenos (CLP). Se incluyen además sus equivalentes aproximados en unidades de fomento (UF) y dólares estadounidenses (USD), con el objetivo de facilitar su interpretación en distintos contextos.

Tabla 5.4: Desglose del costo de implementación del prototipo BioAsist-Device.

Componente	**Precio unitario (CLP)**	**Precio unitario (USD)**	**Precio unitario (UF)**
ESP32 DevKit V1	9.990	10.93	0.25
Sensor de huella R307	10.185	11.14	0.26
Pantalla LCD 16x2 I2C	5.500	6.02	0.14
Powerbank 5V / 10.000 mAh	8.991	9.83	0.23
Total	**34.666**	**37.92**	**0.88**

5.2. Firmware en ESP32

El firmware desarrollado para la ESP32 constituye el núcleo lógico de *BioAsist*, encargado de coordinar la interacción entre el lector de huellas dactilares, la conexión Wi-Fi y el servidor web. Su diseño se basó en la necesidad de mantener un *BioAsist-Device* autónomo, capaz de operar con

recursos limitados, y a la vez suficientemente robusto para comunicarse en tiempo real con la plataforma central.

El firmware del *BioAsist-Device* fue implementado en C++ utilizando el entorno de desarrollo de Arduino para ESP32, sobre el *Arduino core for ESP32* [23] provisto por Espressif. Esta plataforma entrega el modelo de programación basado en las funciones `setup()` y `loop()`, así como las librerías necesarias para gestionar los periféricos del microcontrolador, la comunicación serie con el lector biométrico y la conectividad Wi-Fi empleada para intercambiar información con el servidor PHP de *BioAsist*. De este modo se simplifica el flujo de compilación y carga del programa en la placa, y se reduce la complejidad de la implementación respecto de un desarrollo nativo sobre el SDK de bajo nivel.

5.2.1. Arquitectura general

El programa se estructura en torno a dos grandes ejes: por un lado, el sondeo periódico al servidor para verificar solicitudes de enrolamiento o estados de asistencia, y por otro, la captura y verificación de huellas locales en caso de encontrarse una asistencia activa. Con esta división se busca separar responsabilidades, delegando en el servidor la lógica de cursos y alumnos, y en la ESP32 la operación directa con el hardware biométrico. La interacción con el usuario se canaliza a través de un display LCD 16x2 (interfaz I2C), que entrega mensajes de estado y retroalimentación inmediata durante el proceso [22]. Este comportamiento general se resume en el pseudocódigo presentado en el Algoritmo 2.

5.2.2. Librerías utilizadas

El correcto funcionamiento del firmware depende de un conjunto de librerías cuidadosamente seleccionadas, cada una de las cuales cumple un rol específico:

- **Arduino.h**: librería base que proporciona las funciones esenciales del entorno Arduino sobre ESP32, incluyendo inicialización de periféricos, manejo de pines y temporización. Forma parte del *Arduino core for ESP32* desarrollado por Espressif [23].
- **Wire.h** y **LiquidCrystal_I2C.h**: permiten la comunicación con el LCD 16x2 a través del bus I2C. `Wire.h` establece la comunicación a nivel de protocolo, mientras que `LiquidCrystal_I2C.h` abstrae las operaciones de impresión de texto y control del display. Esta última se encuentra disponible en un repositorio abierto mantenido por la comunidad [22].

  WiFi.h y HTTPClient.h: gestionan la conectividad de red de la ESP32. Con ellas es posible conectar la placa como cliente a una red Wi-Fi con autenticación WPA2-PSK (por ejemplo, una red doméstica o de laboratorio) y enviar solicitudes HTTP GET y POST hacia el servidor PHP de *BioAsist*. Estas librerías forman parte del núcleo desarrollado por Espressif para la plataforma ESP32 [23]. Es importante señalar que la conexión a redes WPA2-Enterprise, como *Eduroam*, requiere el uso de librerías adicionales y configuraciones específicas, que no fueron abordadas en este prototipo.
- **Preferences.h**: otorga acceso a la memoria no volátil (NVS) de la ESP32. En este proyecto se utiliza para almacenar de forma persistente el siguiente identificador de huella disponible (`next_fid`) y los mapeos entre identificadores biométricos y usuarios. La documentación oficial de Espressif detalla su funcionamiento [24].

Algorithm 2: Flujo general del firmware del *BioAsist-Device*

```
Función setup()
    inicializarGPIO();
    inicializarLCD();
    inicializarSensorHuella();
    cargarConfiguracionNVS();
    conectarWiFi();
Función loop()
    estadoServidor ← consultarServidor();
    if estadoServidor indica "enrolamiento solicitado" then
        resultado ← ejecutarEnrolamiento();
        notificarServidor(resultado);
    end
    if estadoServidor indica "asistencia activa" then
        huella ← capturarHuella();
        id ← identificarHuella(huella);
        if id es válido then
            registrarAsistencia(id);
            notificarServidor(id);
            mostrarMensajeLCD(id);
        end
    end
    actualizarPantallaLCD();
    esperarIntervalo();
```

- **Adafruit_Fingerprint.h**: provee las funciones necesarias para interactuar con el sensor de huellas modelo R307, conectado a la UART2. Permite ejecutar operaciones de captura de imagen, generación de características, comparación contra plantillas y almacenamiento en *slots*. Esta librería es mantenida por Adafruit y se encuentra publicada en GitHub [12].
- **map** (STL): se emplea para implementar una estructura de control de tiempos de espera por usuario, evitando que un mismo alumno pueda registrar múltiples asistencias consecutivas en intervalos muy cortos.

Una decisión de diseño relevante fue evitar el uso de un parser JSON completo, lo que habría incrementado el consumo de memoria y dependencias. En su lugar se optó por realizar un procesamiento manual de cadenas, suficiente para los mensajes simples que entrega el servidor, aunque menos robusto frente a variaciones de formato.

5.2.3. Persistencia local

El sistema mantiene información esencial en la memoria no volátil de la ESP32 mediante la librería `Preferences.h` [24]. Se utiliza un *namespace* denominado `elo308-map` que contiene dos tipos de registros: la clave `next_fid`, que indica el próximo identificador de huella disponible, y las claves individuales de la forma `f_<finger_id>`, que almacenan el identificador de usuario asociado a cada plantilla biométrica. De este modo se garantiza la coherencia entre el sensor y el servidor incluso tras reinicios del *BioAsist-Device*.

5.2.4. Flujo de arranque

La función `setup()` inicializa los distintos módulos en el siguiente orden:

1. Inicialización de los periféricos básicos y la activación del bus I2C para el display LCD.
2. Carga de los parámetros persistentes almacenados en la memoria no volátil (NVS), incluyendo el identificador de huella disponible.
3. Conexión a la red Wi-Fi mediante la función `wifiConnect()` A.5.1.
4. Establecimiento de la comunicación con el sensor de huellas, verificación de contraseña y lectura de parámetros.
5. Ejecución opcional de borrado completo de base local y del sensor si está activa la bandera de administración.

5.2.5. Bucle principal de operación

En la función `loop()` se concentra la lógica repetitiva del sistema:

1. Sondeo de solicitudes de enrolamiento con `pollForEnrollment()` A.2.1.
2. Ejecución del proceso de enrolamiento automático mediante `doEnrollmentAuto()` A.2.2 en caso de existir solicitud.
3. Verificación del estado de asistencia a través de `pollAttendanceState()` A.3.1.
4. Activación de la captura y registro de huellas con `scanAndRegisterAttendance()` A.3.2 cuando la asistencia está activa.

5.2.6. Proceso de enrolamiento

El enrolamiento de un nuevo usuario sigue una secuencia guiada que asegura la correcta creación de la plantilla biométrica:

1. Recepción del `id_usuario` desde el servidor.
2. Doble captura de huella mediante las funciones `getImage()` e `image2Tz()`.
3. Creación de la plantilla biométrica final con `createModel()` y almacenamiento en un *slot* libre con `storeModel()`.
4. Actualización de la memoria NVS a través de `saveFingerUserMap()` A.4.1.
5. Confirmación del enrolamiento al servidor mediante `httpPostForm()` A.5.2.

5.2.7. Proceso de asistencia

Cuando existe una asistencia activa, el *BioAsist-Device* ejecuta el siguiente flujo:

1. Captura de la huella con `getImage()` y conversión a características.
2. Búsqueda en la base de plantillas mediante `fingerSearch()`.

3. Obtención del identificador de usuario en NVS con `getUserByFinger()` A.4.2.
4. Aplicación del *cooldown* controlado por un `map` que registra el último pase de cada usuario.
5. Envío del registro de asistencia al servidor mediante `httpPostForm()` A.5.2.

5.2.8. Decisiones de implementación

La implementación del firmware privilegió la simplicidad y la robustez. Se optó por una persistencia mínima, limitándose a mantener el mapeo huella–usuario y el contador del próximo identificador de plantilla biométrica. La lógica compleja, como la gestión de ramos, alumnos y validaciones adicionales, se trasladó al servidor, reduciendo así la carga en la ESP32. Por otra parte, se consideró la tolerancia a fallos de red: ante una caída temporal de conectividad, el estado de enrolamiento se mantiene como "pendiente" para evitar la interrupción del proceso de enrolamiento. La interfaz con el usuario también fue priorizada, con mensajes claros en cada paso del proceso mediante el LCD (I2C), lo que facilita la usabilidad en un entorno real de clases.

5.3. Backend PHP / API REST: endpoints y lógica

El backend implementado en PHP expone un conjunto de endpoints de tipo REST (ligeros) que permiten orquestar el enrolamiento de huellas, la activación de asistencias por ramo, el registro de asistencias individuales y la consulta de estados por parte del firmware en la ESP32. La API opera sobre una base de datos MySQL y encapsula la lógica de negocio (validaciones, consistencia por fecha/ramo, unicidad de registros y generación de ausentes), manteniendo al *BioAsist-Device* como un cliente simple que ejecuta las instrucciones del servidor de *BioAsist*.

A nivel estructural, el servidor web se compone de un conjunto de vistas y scripts PHP organizados según el rol del usuario (administrador, profesor y alumno), además de una serie de endpoints destinados exclusivamente a la comunicación con el *BioAsist-Device*. Para facilitar la comprensión de este módulo, en la Figura **??** se presenta un diagrama que resume las dependencias entre los distintos archivos PHP que conforman la plataforma web.

El diagrama anterior presenta el flujo jerárquico de archivos comenzando desde `login.php`, punto de acceso común para los tres tipos de usuario definidos en la plataforma. Desde allí se desprenden las vistas específicas de cada perfil: el administrador, encargado de la creación de usuarios y la gestión del proceso de enrolamiento; el docente, responsable de la creación de ramos, la asociación de alumnos y la administración de sesiones de asistencia; y el estudiante, quien accede únicamente a su historial de asistencias. Cada vista se apoya en scripts adicionales que procesan solicitudes particulares, ya sea mediante formularios tradicionales o mediante consultas asíncronas (AJAX), lo que permite mantener una interfaz fluida y modular.

Además de las vistas y scripts mostrados en este mapa, el backend incorpora un conjunto de rutas que procesan acciones enviadas desde la interfaz web (como la activación de asistencias, la creación de ramos o la gestión de solicitudes de enrolamiento). Estas rutas se describen en la subsección siguiente, antes de presentar los endpoints REST utilizados exclusivamente por el *BioAsist-Device* durante los procesos de enrolamiento y registro de asistencia.

5.3.1. Acciones de panel

Estas rutas procesan formularios enviados desde la interfaz web y realizan cambios en la base de datos. No devuelven JSON, sino que ejecutan la lógica en servidor y luego redirigen a vistas específicas para mostrar el resultado al usuario.

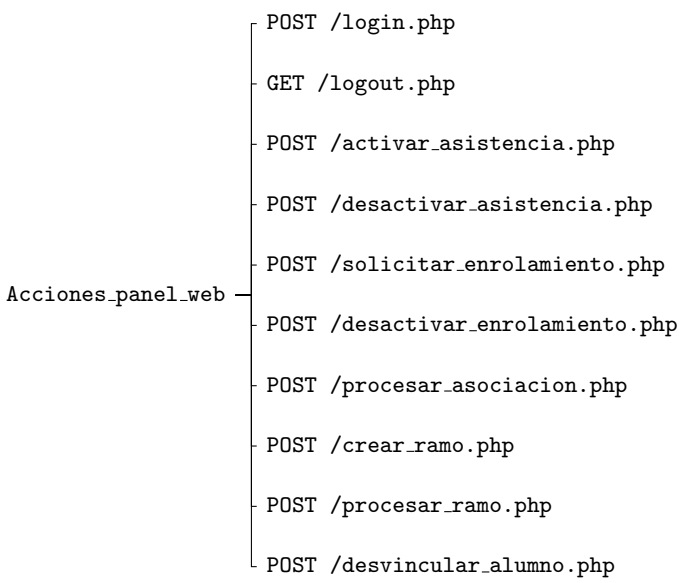


Figura 5.8: Endpoints de acción invocados desde el panel web (autenticación y operaciones sobre asistencia, enrolamiento y ramos).

- **POST** `/activar_asistencia.php`: activa la asistencia del día para un `id_ramo`, estableciendo `activo=1` y `hora_inicio=NOW()`. Tras la operación redirige a `/pasar_asistencia.php`, donde el docente puede visualizar en vivo a los alumnos que marcan asistencia.
- **POST** `/desactivar_asistencia.php`: cierra la asistencia de un ramo (`activo=0`, `hora_fin`) y, si corresponde, inserta automáticamente registros de ausentes para quienes no marcaron. Luego redirige a `/asistencia_profesores.php`, que muestra el resumen de la sesión recién cerrada.
- **POST** `/solicitar_enrolamiento.php`: genera una nueva solicitud en la tabla `enrolamiento` para un `id_usuario`. Redirige a `/enrolar.php`, donde se monitorea el progreso del enrolamiento.
- **POST** `/desactivar_enrolamiento.php`: cancela una solicitud de enrolamiento activa para un `id_usuario`. También redirige a `/enrolar.php`, mostrando el estado actualizado.
- **POST** `/procesar_asociacion.php`: recibe un conjunto de `id_alumnos[]` y un `id_ramo`, y vincula esos alumnos al ramo en la tabla `alumnos_ramos`. Redirige a `/ver_alumnos_ramo.php`.
- **POST** `/crear_ramo.php`: crea un nuevo ramo en la tabla `ramos`. Redirige a `/gestionar_ramos.php`.
- **POST** `/procesar_ramo.php`: permite editar el nombre de un ramo o eliminarlo si no existen asistencias asociadas. Redirige nuevamente a `/gestionar_ramos.php`.
- **POST** `/desvincular_alumno.php`: elimina la relación entre un alumno y un ramo en la tabla `alumnos_ramos`. Redirige a `/ver_alumnos_ramo.php`.
- **POST** `/login.php`: valida credenciales e inicia sesión según el rol del usuario.
- **GET** `/logout.php`: destruye la sesión activa y redirige a `/login.php`.

Como complemento, el mapa completo de vistas y scripts del backend web se presenta en el Anexo B.1 (Fig. B.1).

5.3.2. Endpoints auxiliares del panel web

Consultas y reportes (AJAX/descarga)

- **GET** `/obtener_asistencia_ajax.php`: utilizado por las vistas del panel del profesor para mostrar en tiempo real la lista de alumnos presentes. Recibe `id_ramo` como parámetro, consulta la tabla `asistencia` filtrando por ramo y fecha actual, y realiza un `JOIN` con `usuarios` para recuperar el nombre completo de cada alumno. La respuesta es un JSON con un arreglo de alumnos presentes, que se consume mediante peticiones AJAX.
- **GET** `/obtener_alumnos_disponibles.php`: entrega en formato JSON los alumnos que pueden asociarse a un ramo en la interfaz de administración. Dado un `id_ramo`, el endpoint consulta los alumnos de la tabla `usuarios` con tipo `'alumno'` y excluye aquellos que ya estén inscritos en la tabla `alumnos_ramos` para ese ramo, construyendo la lista de candidatos disponibles.
- **GET** `/descargar_asistencia.php`: genera un archivo CSV con los registros de asistencia de un ramo específico, filtrados por rango temporal (día, semana o mes) según los parámetros `filtro` y `fecha`. El endpoint verifica primero que el ramo solicitado pertenezca al profesor autenticado en sesión y luego ejecuta una consulta que une las tablas `asistencia` y `usuarios`, ordenando los resultados por fecha y nombre. El archivo resultante contiene, para cada registro, la fecha, nombre del alumno, rol y estado de asistencia, y está pensado para su uso en planillas de cálculo o importación en sistemas externos.

5.3.3. API del *BioAsist-Device*

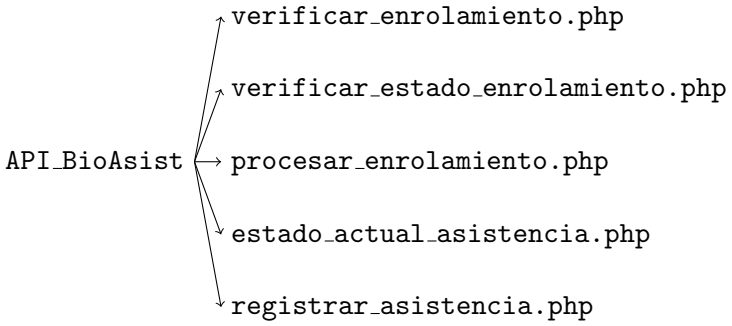


Figura 5.9: Endpoints REST consumidos por el *BioAsist-Device*.

Los endpoints mostrados en la Figura 5.9 conforman la interfaz REST utilizada por el *BioAsist-Device* para coordinar el proceso de enrolamiento y para registrar asistencias durante una sesión activa. Estos endpoints retornan objetos JSON y están diseñados para operar de forma idempotente y tolerante a fallos de conectividad, de modo que el dispositivo pueda consultar periódicamente el estado del sistema sin generar duplicaciones o inconsistencias.

1) Enrolamiento de usuarios (*BioAsist-Device*)

- **GET** `/verificar_enrolamiento.php`: es consultado periódicamente por el *BioAsist-Device* para verificar si existe una solicitud de enrolamiento pendiente en la tabla `enrolamientos`. Internamente, el endpoint realiza una consulta que selecciona el primer registro con estado `'pendiente'` y retorna, en caso de existir, un objeto JSON con el identificador del enrolamiento (`id_enrolamiento`) y el identificador del usuario a enrolar (`id_usuario`). Si no hay solicitudes, responde con un mensaje indicando que no existen tareas pendientes. Esta respuesta activa el inicio del flujo de doble captura de huella en el firmware.
- **GET** `/verificar_estado_enrolamiento.php`: dado un `id_usuario`, entrega el estado de la solicitud de enrolamiento asociada (por ejemplo, `pendiente`, `cancelado` o `confirmado`). El endpoint obtiene el identificador desde el parámetro *`GET`* `id_usuario` y consulta la tabla `enrolamientos` para recuperar el estado más reciente. Esta información permite al firmware cerrar el flujo de enrolamiento local (limpiar memoria o abortar el proceso) cuando la solicitud es resuelta desde la interfaz web.
- **POST** `/procesar_enrolamiento.php`: es llamado por el *BioAsist-Device* al completar la captura biométrica. Recibe vía `POST` el `id_usuario` y el `finger_id` asignado por el sensor, y ejecuta tres acciones principales: (i) valida que exista un enrolamiento pendiente para ese usuario, (ii) actualiza la tabla `usuarios` registrando el nuevo `finger_id` asociado, y (iii) marca el registro correspondiente en `enrolamientos` como confirmado, guardando el resultado para auditoría. La respuesta JSON indica el éxito de la operación o un mensaje de error en caso de no existir una solicitud pendiente.

2) Control de asistencia (*BioAsist-Device*)

- **GET** `/estado_actual_asistencia.php`: consulta si existe una asistencia activa para la fecha actual. El endpoint busca en la tabla `estado_asistencia` un registro con `activo = 1` para el día en curso y, de existir, retorna un JSON con `activo = true` y el identificador del ramo asociado (`id_ramo`, `paralelo`). En caso contrario, responde `activo = false`. El *BioAsist-Device* almacena temporalmente este `id_ramo` y lo reutiliza en cada registro de asistencia mientras la sesión permanezca activa.
- **POST** `/registrar_asistencia.php`: registra la asistencia de un alumno para un ramo activo. El dispositivo envía al servidor el `id_usuario` identificado localmente a partir de la huella (usando el mapeo huella–usuario almacenado en NVS) y el `id_ramo` obtenido previamente desde `/estado_actual_asistencia.php`. El endpoint verifica que efectivamente exista una asistencia activa para ese ramo y que no se haya registrado ya una asistencia para la combinación `(id_alumno, id_ramo, fecha)`. Si la validación es exitosa, inserta un nuevo registro en la tabla `asistencia` con la fecha y hora del servidor y responde con `ok = true` y un código de mensaje (por ejemplo, `ASISTENCIA_REGISTRADA`). En caso de asistencia inactiva o intento duplicado, responde con `ok = false` y un `error_code` específico, garantizando la idempotencia incluso si existen múltiples *BioAsist-Device* reportando al mismo servidor.

5.3.4. Contratos de respuesta y manejo de errores

Todas las respuestas devuelven JSON con una clave `ok` booleana y, en caso de error, `error_code` y un mensaje explicativo. La API utiliza `HTTP 200` para errores de negocio previsibles (por simplicidad de parsing en firmware) y reserva `HTTP 5xx` para fallas del servidor. Ejemplos:

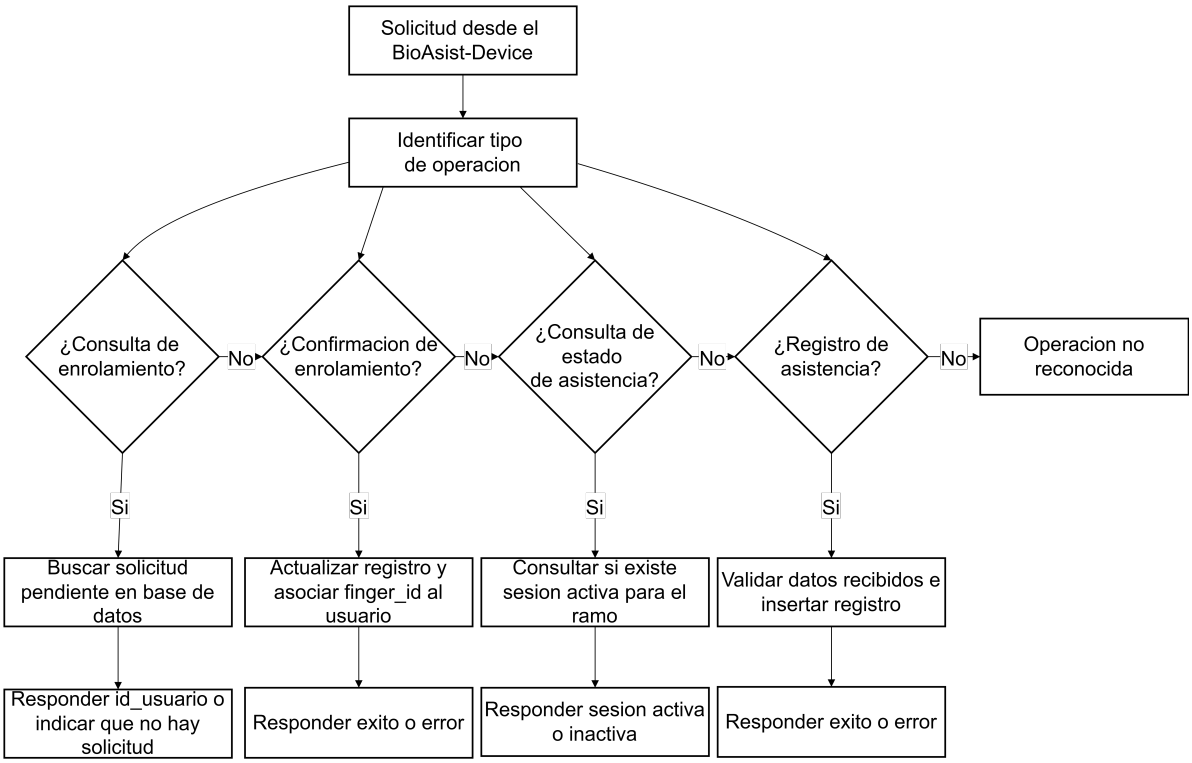


Figura 5.10: Flujo de procesamiento de solicitudes en el backend del sistema BioAsist.

```
{ "ok": false, "error_code": "ASIS_INACTIVA", "msg": "No hay asistencia activa" }
{ "ok": false, "error_code": "DUPLICADO",     "msg": "Ya existe registro para hoy" }
```

La idempotencia en `/registrar_asistencia.php` se garantiza por la restricción única (id_alumno, id_ramo, fecha); los reintentos del firmware reciben `ok=true` con el registro existente.

5.3.5. Flujos críticos de negocio

Para complementar la definición de los endpoints, la Figura 5.10 presenta el flujo general de procesamiento que realiza el backend ante las solicitudes enviadas por el BioAsist-Device. Este diagrama resume la lógica de decisión principal: identificar el tipo de operación, consultar o actualizar la base de datos según corresponda y generar la respuesta apropiada. La estructura mostrada refleja el comportamiento real del servidor y permite visualizar cómo se orquestan las distintas operaciones en el sistema.

Enrolamiento guiado por servidor

El proceso de enrolamiento se coordina entre el panel web, el servidor y el *BioAsist-Device*. El panel genera una solicitud para un `id_usuario`, la cual queda registrada en la tabla `enrolamientos`. El dispositivo consulta periódicamente al servidor para detectar solicitudes pendientes; cuando el servidor retorna un `id_usuario`, el dispositivo inicia el proceso de captura biométrica local.

Una vez completada la captura, el *BioAsist-Device* reporta al servidor el `finger_id` generado, y el backend persiste la asociación en la tabla `usuarios`, marcando la solicitud como resuelta. Si la solicitud es cancelada desde el panel antes de completar el proceso, el servidor informa este estado en la siguiente consulta y el dispositivo aborta la operación de forma segura.

Registro de asistencia controlado por estado

El inicio y término de un período de asistencia se controla desde el panel web. Cuando un profesor activa la asistencia de un ramo, el servidor marca dicho ramo como `activo`. El *BioAsist-Device* consulta periódicamente el estado mediante el endpoint correspondiente y, al detectar una sesión activa, entra en modo de escaneo.

Ante una coincidencia de huella, el dispositivo envía al servidor el `id_usuario` identificado localmente y el `id_ramo` informado en la activación. El backend valida que la asistencia esté activa, aplica idempotencia para evitar registros duplicados y almacena la hora del servidor asociada al registro. Al finalizar el período, el panel invoca el endpoint de cierre, y el servidor marca la asistencia como concluida y registra ausencias cuando corresponde.

5.3.6. Consideraciones de implementación del backend

El backend privilegia la claridad de contratos (JSON simple y estable), la idempotencia en operaciones sensibles (registro de asistencia) y la coherencia temporal anclada a la hora del servidor. Los endpoints expuestos al *BioAsist-Device* son deliberadamente *estrechos*: sólo los necesarios para enrolar y marcar asistencia. La mayor parte de la lógica administrativa (alta/baja de ramos, asociación de alumnos, reportes, CSV) queda en endpoints separados usados por el panel web. Esta compartimentación permite evolucionar la interfaz de usuario sin afectar al firmware y facilita el versionado de la API.

5.3.7. Referencias al Apéndice

Para mantener el capítulo legible, los listados completos de código se incluyen en el Apéndice A. En particular:

- `verificar_enrolamiento.php` (*listado* A.7.1) y `verificar_estado_enrolamiento.php` (*listado* A.7.3).
- `procesar_enrolamiento.php` (*listado* A.7.2).
- `estado_actual_asistencia.php` (*listado* A.7.4).
- `registrar_asistencia.php` (*listado* A.7.5).
- `activar_asistencia.php` y `finalizar_asistencia.php` (*listado* A.7.6).

5.4. Interfaz web: HTML/CSS/JS y funcionalidades

La interfaz web constituye el punto de contacto principal entre usuarios humanos (profesores, alumnos y administradores) y *BioAsist*. Fue desarrollada en `PHP` para la lógica de servidor y generación dinámica de vistas, con `HTML5` y `CSS3` para la estructura y el diseño, y complementada por `JavaScript` para interacciones dinámicas (AJAX). La organización modular del backend permite que cada vista invoque únicamente los endpoints que requiere, manteniendo un frontend ligero y funcional, capaz de adaptarse a los distintos perfiles de usuario.

5.4.1. Clasificación de vistas por tipo de usuario

Las vistas están divididas según el rol de usuario:

- **Administrador**: accede a un panel de control global donde puede crear cuentas (profesores o alumnos), gestionar ramos y supervisar procesos de enrolamiento.
- **Profesor**: dispone de vistas específicas para enrolar a los alumnos, pasar asistencia de sus ramos, crear/editar/borrar ramos, inscribir alumnos y monitorear en vivo a los presentes, además de generar reportes históricos.
- **Alumno**: accede a su propio historial de asistencias de los ramos en los que está inscrito, con una interfaz simplificada y de fácil consulta.

Cada vista combina plantillas PHP que generan HTML dinámico con hojas de estilo CSS para mantener una apariencia consistente en todo el sistema. El control de acceso por rol asegura que las funcionalidades se desplieguen únicamente al tipo de usuario autorizado.

5.4.2. Tecnologías utilizadas

El diseño de la interfaz web se basó en un conjunto de tecnologías ampliamente aceptadas en el desarrollo de aplicaciones en línea. La elección buscó un equilibrio entre simplicidad de implementación, compatibilidad con distintos navegadores y facilidad de mantenimiento futuro.

- **HTML5**: lenguaje base de la interfaz, encargado de definir la estructura semántica de las páginas. Se utiliza para organizar formularios (login, creación de ramos, asociación de alumnos), tablas de asistencia y elementos de navegación. El uso de etiquetas semánticas (como `<header>`, `<section>` y `<table>`) favorece la accesibilidad y la comprensión.
- **CSS3**: capa de estilo y presentación visual. Mantiene coherencia en la apariencia de botones, formularios y tablas, e implementa diseño responsivo mediante *flexbox* y *grid*, permitiendo uso en escritorio y móviles.
- **JavaScript**: aporta dinamismo en el cliente. Valida formularios (por ejemplo, formato del correo), actualiza listados de manera automática y gestiona interacciones inmediatas como la creación o cancelación de solicitudes de enrolamiento, evitando recargas innecesarias.
- **AJAX (Asynchronous JavaScript and XML)**: implementado con `XMLHttpRequest`, intercambia datos con el servidor sin recargar la página completa. Se utiliza para el refresco en vivo de asistencia (`/obtener_asistencia_ajax.php`) y la verificación de estado de enrolamiento, de modo que los profesores visualicen en tiempo real a los alumnos presentes y los administradores monitoreen procesos sin intervención adicional.

En conjunto, estas tecnologías conforman una arquitectura web ligera, sin dependencias en frameworks externos, lo que facilita la comprensión y el mantenimiento por futuros desarrolladores. A pesar de su simplicidad, cumplen adecuadamente con los requisitos de interactividad, consistencia visual y soporte multiplataforma.

5.4.3. Vistas principales

- `/admin_panel.php`: tablero principal para administradores, con accesos a creación de cuentas (profesores y alumnos). Requiere sesión con rol de administrador y actúa como *hub* de mantenimiento de *BioAsist*.
- `/gestionar_ramos.php`: lista, renombra o elimina ramos del profesor autenticado. Los formularios se procesan en servidor mediante `/procesar_ramo.php` (POST) siguiendo el patrón *Post/Redirect/Get* (PRG) para volver a esta vista con el estado actualizado.
- `/ver_alumnos_ramo.php`: presenta los alumnos inscritos en un ramo específico (parámetro `id_ramo`) y ofrece desvinculación individual. Tras cada operación, redirige a la misma vista para reflejar la lista vigente.
- `/asociar_alumnos.php`: interfaz para vincular múltiples alumnos a un ramo; utiliza `/obtener_alumnos_disponibles.php` (AJAX) para poblar opciones dinámicamente. El envío del formulario se procesa vía `/procesar_asociacion.php` (POST) y redirige a `/ver_alumnos_ramo.php` con el resultado.
- `/pasar_asistencia.php`: vista operativa para clases en curso; muestra en vivo los presentes consultando periódicamente `/obtener_asistencia_ajax.php` (AJAX). Está pensada para usarse tras activar la asistencia del día y evita recargas completas para una experiencia fluida.
- `/asistencia_profesores.php`: genera reportes por ramo con filtros de fecha/rango y métricas de presentes/ausentes. Ofrece exportación directa a CSV mediante `/descargar_asistencia.php`, facilitando uso en planillas o LMS.
- `/asistencia_alumno.php`: muestra al alumno autenticado su historial de asistencias, ordenado por fecha y ramo. Proporciona una visualización simple para seguimiento personal sin funcionalidades administrativas.
- `/enrolar.php`: gestiona la creación y cancelación de solicitudes de enrolamiento (formularios procesados por `/solicitar_enrolamiento.php` y `/desactivar_enrolamiento.php`). La vista permite monitorear el estado mientras el *BioAsist-Device* ejecuta la captura y confirmación.
- `/ver_asistencia_tiempo_real.php`: actualiza en intervalos cortos la lista de alumnos que marcan asistencia, reutilizando `/obtener_asistencia_ajax.php`. Prioriza legibilidad (fuente grande, lista limpia) para una posible visualización pública.

5.4.4. Funcionalidades dinámicas

- **Actualización en vivo**: llamadas periódicas a `/obtener_asistencia_ajax.php` refrescan automáticamente las listas de alumnos presentes en `/pasar_asistencia.php` y `/ver_asistencia_tiempo_real.php`.
- **Manejo de formularios**: las operaciones críticas (crear ramo, activar asistencia, solicitar enrolamiento) se ejecutan vía POST y luego redirigen, siguiendo *Post/Redirect/Get* para evitar reenvíos accidentales y duplicados.

- **Validación en cliente**: JavaScript valida entradas básicas (campos obligatorios, formato de correo) antes de enviar formularios, reduciendo errores y mejorando la experiencia.
- **Consistencia visual**: el CSS asegura estilo uniforme en formularios, tablas y botones, reforzando la usabilidad y la percepción de sistema integrado.

La interfaz web complementa el firmware y la API al proveer un medio intuitivo para administrar y monitorear *BioAsist*. Su simplicidad tecnológica (PHP, HTML/CSS estándar y JavaScript básico) reduce la curva de aprendizaje y facilita el mantenimiento, sin sacrificar la interactividad necesaria para un uso fluido en el contexto académico. Esta capa de presentación garantiza que las funcionalidades desarrolladas en el backend sean accesibles y comprensibles para los distintos perfiles de usuario.

5.5. Implementacion avanzada: Prueba de concepto ESP-IDF

Si bien el núcleo operativo de BioAsist-Device se desarrolló sobre el framework de Arduino por su agilidad en el manejo de stacks de red, se identificó una limitación crítica en las librerías comerciales: la opacidad en el manejo de los paquetes de datos biométricos. Para resolver esto, se desarrolló una prueba de concepto utilizando ESP-IDF (Espressif IoT Development Framework), con el objetivo de validar la extracción y recarga manual de plantillas (templates).

5.5.1. Control del Protocolo a Bajo Nivel

A diferencia de las implementaciones estándar, el uso de ESP-IDF permitió interactuar directamente con la capa de enlace del sensor R307 a través del bus UART2. Se refactorizaron los drivers de comunicación para gestionar manualmente los paquetes de datos, específicamente las instrucciones de carga y descarga de modelos:

- **Instrucción 08H (Upload Template):** Permite extraer la plantilla generada en el buffer del sensor hacia la memoria del microcontrolador.
- **Instrucción 09H (Download Template):** Permite enviar una plantilla previamente almacenada desde el microcontrolador hacia el buffer del sensor para su comparación.

5.5.2. Resultados del Hito Técnico

La prueba de concepto logró completar con éxito el ciclo de persistencia externa:

- Enrolamiento de una huella y generación de la plantilla en el R307.
- Extracción exitosa de los datos binarios hacia el ESP32 mediante el comando 08H.
- Vaciado completo de la base de datos local del sensor.
- Recarga de la plantilla desde el ESP32 al sensor mediante el comando 09H y verificación de identidad exitosa (Match).

Este resultado, aunque no se migró al sistema de producción final por razones de tiempo y compatibilidad con el resto de los servicios HTTP, constituye una validación fundamental. Demuestra que la arquitectura de BioAsist es capaz de soportar una sincronización bidireccional, permitiendo que un alumno enrolado en un dispositivo pueda ser reconocido en cualquier otro nodo de la red universitaria mediante la itinerancia de sus plantillas biométricas.

Capítulo 6

Verificación y Validación

La etapa de verificación y validación de *BioAsist* tuvo como propósito garantizar que cada uno de los módulos de hardware y software cumpliera con su función de manera independiente, y posteriormente validar el desempeño del sistema completo bajo condiciones de uso reales. Para ello, el proceso se dividió en tres fases: (i) pruebas funcionales por módulo, (ii) pruebas del sistema integrado y (iii) evaluación de desempeño.

6.1. Pruebas funcionales por módulo

En esta primera fase se realizaron pruebas unitarias, centradas en comprobar el funcionamiento de cada componente de forma aislada antes de proceder a la integración. Para este fin se utilizaron principalmente códigos de ejemplo provistos por las bibliotecas oficiales [12, 22, 25], que luego fueron adaptados al contexto del proyecto. Los códigos empleados para estas pruebas se encuentran recopilados en el *Apéndice A*, de modo que se mantiene un registro reproducible de los ensayos realizados.

6.1.1. Lector biométrico

El sensor R307 fue probado mediante la librería *Adafruit Fingerprint* [12], que incluye ejemplos básicos de enrolamiento e identificación de huellas.

Para estas pruebas se utilizó exactamente la misma configuración de hardware presentada en el Capítulo 5.1, específicamente las conexiones del lector R307 hacia la ESP32 mostradas en la Figura 5.3 y en la fotografía anotada de la Figura 5.4b. El sensor se conectó a la placa *ESP32 DevKit V1* mediante la interfaz UART2 (TXD–RXD), con alimentación de 5 V suministrada por el propio ESP32 a través de su pin VIN.

Es importante destacar que el computador no interactuó directamente con el sensor; su rol se limitó a visualizar el monitor serial mientras el código era ejecutado íntegramente en la ESP32, tal como se describe en el montaje físico del Capítulo 5.

En primer lugar, se ejecutó el código de enrolamiento. Este guiaba al usuario a colocar el dedo dos veces para registrar la huella en la memoria interna del módulo. Los mensajes de estado y confirmación eran mostrados en el monitor serial, lo que permitió verificar paso a paso el proceso (Figura 6.1).

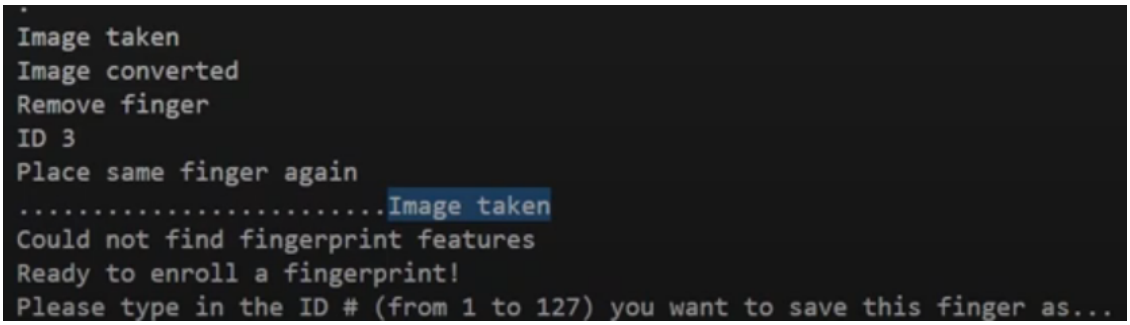


Figura 6.1: Verificación de funcionamiento de la función de enrolar del lector R307

Posteriormente, se evaluó la función de identificación del sensor, utilizando la misma configuración de hardware empleada durante el enrolamiento y descrita previamente (ver Figuras 5.3 y 5.4b). Al ejecutar el ejemplo de identificación provisto por la librería *Adafruit Fingerprint*, el sistema adquirió una nueva imagen de huella, la convirtió a sus características internas y la comparó contra la base de datos de plantillas almacenadas en el lector.

Como se muestra en la Figura 6.2, el módulo fue capaz de reconocer correctamente una huella registrada con anterioridad, indicando el número de ID asociado y el nivel de confianza obtenido.

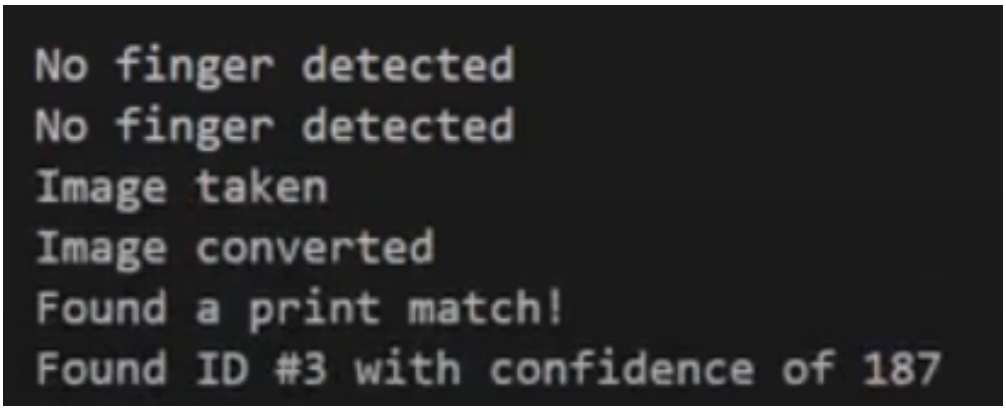


Figura 6.2: Verificación de funcionamiento de la función de identificación del lector R307

Estos resultados confirmaron que el sensor es capaz de registrar e identificar huellas con un nivel adecuado de confiabilidad, y que la comunicación UART con la ESP32 se mantuvo estable durante todo el proceso, sin pérdida de datos.

6.1.2. Display LCD

Para el display LCD 16x2 con interfaz I2C se empleó la librería *LiquidCrystal_I2C* [22]. La conexión utilizada en estas pruebas corresponde exactamente a la mostrada en el Capítulo 5.1, donde se detalla el cableado entre la pantalla y la *ESP32 DevKit V1* (Figura 5.6) y se presenta la fotografía anotada del módulo con su pinout (Figura 5.5).

El display se conectó a los pines SDA y SCL de la interfaz I2C de la ESP32 (GPIO 21 y GPIO 22, respectivamente), mientras que su alimentación se realizó mediante los 5 V del propio módulo, tal como se describe en la sección de hardware del Capítulo 5. El computador no interactuó directamente con el display: su rol se limitó a cargar el programa y visualizar mensajes de depuración mediante el monitor serial.

Mediante un código de prueba se inicializó el periférico y se enviaron distintos mensajes a ambas líneas de la pantalla, con el fin de verificar su dirección I2C y el correcto funcionamiento del módulo (Figura 6.3).

Figura 6.3: Verificación de dirección y conexión del display LCD I2C.

Los resultados permitieron validar la dirección I2C del controlador del display y confirmar que la alimentación proporcionada por el ESP32 era suficiente para un funcionamiento estable. Esta prueba también permitió verificar que el *BioAsist-Device* contaba con una interfaz visual independiente del monitor serial, elemento fundamental para la interacción con profesores y alumnos en el dispositivo final.

6.1.3. Conexión Wi-Fi

Finalmente, se validó la capacidad de la ESP32 de conectarse a una red inalámbrica mediante un ejemplo provisto por la documentación oficial del entorno Arduino para ESP32 [25]. En este caso, el código establecía una conexión Wi-Fi y creaba un servidor web sencillo, desde el cual era posible encender y apagar el LED integrado de la placa utilizando un navegador web.

La prueba se realizó empleando una placa *ESP32 DevKit V1* conectada al computador únicamente mediante el puerto USB de programación, el cual proporcionó alimentación y permitió visualizar los mensajes del monitor serial. La comunicación inalámbrica se efectuó a través del módulo Wi-Fi integrado del ESP32, configurado para conectarse a una red local de 2.4 GHz. El computador y el ESP32 estuvieron conectados a la misma red para permitir el acceso al servidor web generado por el ejemplo.

Esta prueba permitió comprobar no solo la capacidad del ESP32 de integrarse a una red Wi-Fi, sino también su habilidad para recibir y procesar peticiones HTTP en tiempo real. Con ello se verificó que la plataforma era apta para implementar, en etapas posteriores, la comunicación con el servidor remoto del sistema *BioAsist*.

6.2. Pruebas del sistema integrado

Una vez validados los módulos de forma aislada, se procedió a su integración en un sistema único. Para ello, se tomaron los códigos de enrolamiento y de toma de asistencia —que en la versión inicial comunicaban información exclusivamente por puerto serial— y se adaptaron para mostrar los mensajes en el display LCD y comunicarse con el servidor mediante Wi-Fi.

La configuración de hardware utilizada en esta etapa corresponde exactamente a la descrita en el Capítulo 5.1, donde se presentan los esquemas individuales de conexión del sensor R307, la pantalla LCD y la ESP32, así como el diagrama completo del sistema integrado (Figura **??**). Dicho montaje incluyó una placa *ESP32 DevKit V1*, el sensor biométrico R307 conectado por UART, un display LCD 16x2 mediante I2C y una batería portátil de 5V como fuente de alimentación.

El firmware empleado correspondió a la primera versión completamente integrada, incluyendo las funciones de enrolamiento, verificación de huella y envío de información hacia la API REST del servidor. Tanto el dispositivo como el computador servidor se conectaron a la misma red Wi-Fi para permitir la comunicación mediante solicitudes HTTP.

Para validar la solución se realizó una prueba con un curso de 13 estudiantes. En dos sesiones distintas se utilizó el sistema para registrar la asistencia completa del grupo. Durante estas pruebas se midieron:

- El tiempo total necesario para completar el registro de todos los estudiantes (medido mediante cronómetro externo).
- La cantidad de intentos fallidos de lectura de huella, ya sea por error de reconocimiento o por colocación incorrecta del dedo.

Los mensajes desplegados en el LCD permitieron mejorar la experiencia del usuario frente a errores, en comparación con la versión inicial basada únicamente en el monitor serial. Además, la integración con la API confirmó que el dispositivo era capaz de registrar asistencia en tiempo real en la base de datos del sistema, validando así el funcionamiento conjunto de los módulos de captura biométrica, interfaz de usuario y comunicación de red.

6.3. Evaluación de desempeño

La evaluación final de *BioAsist* tuvo como objetivo analizar tanto los tiempos de operación como la confiabilidad del proceso de registro de asistencia, considerando condiciones reales de uso en un curso de tamaño medio.

En primer lugar, respecto al tiempo de operación, se realizaron dos mediciones con un curso de 13 estudiantes. En la primera pasada, el tiempo total para completar el registro fue de 2 minutos con 17 segundos, mientras que en la segunda se redujo a 1 minuto con 42 segundos. Esto corresponde a un promedio de aproximadamente **1 minuto con 59 segundos**, lo que equivale a un tiempo promedio de **9,2 segundos por estudiante**. Este valor se considera plenamente aceptable para el contexto académico, ya que permite registrar una asistencia completa sin afectar el inicio de la clase.

Durante la primera medición se presentaron 5 fallos de lectura, mientras que en la segunda únicamente 2, lo cual evidencia una mejora atribuible a la familiarización de los usuarios con la colocación adecuada del dedo sobre el sensor. Estos fallos no supusieron un inconveniente crítico, pues en todos los casos bastó repetir la autenticación para completar exitosamente el registro.

En segundo lugar, el análisis de confiabilidad mostró que el número de errores de reconocimiento fue muy bajo en relación con el total de intentos. Dichos errores se asociaron principalmente a un posicionamiento incorrecto del dedo en el sensor, situación esperable en una primera interacción de los estudiantes con el *BioAsist-Device*. La repetición inmediata de la lectura permitió superar estas incidencias sin afectar de forma significativa el flujo de la clase.

En cuanto a la infraestructura de red, la conexión Wi-Fi y la comunicación entre el dispositivo y el servidor resultaron estables durante las sesiones de prueba, sin detectarse desconexiones a la

red Wi-Fi, retrasos perceptibles ni pérdida de datos. Este aspecto es relevante, ya que asegura la disponibilidad de la información en tiempo real para su consulta por parte del docente.

En síntesis, los resultados obtenidos permiten concluir que *BioAsist* cumple con los requisitos planteados: asegura la rapidez del proceso de registro, mantiene una confiabilidad adecuada tanto en el reconocimiento biométrico como en la estabilidad de la comunicación, y mejora de forma significativa la experiencia de uso gracias a la interfaz visual del display. La reducción de los tiempos de espera y la automatización del registro en la base de datos contribuyen a un sistema robusto y aplicable a cursos de mayor tamaño y sesiones de asistencia recurrentes.

Capítulo 7

Conclusiones y Trabajo Futuro

El desarrollo de *BioAsist* permitió abordar de manera integral el diseño, implementación y validación de un sistema biométrico de registro de asistencia orientado a entornos académicos. En este capítulo se presentan, en primer lugar, las conclusiones derivadas del proceso de diseño y de las pruebas realizadas, destacando los aspectos del sistema que alcanzaron un nivel satisfactorio de funcionamiento. Posteriormente, se discuten las oportunidades de mejora y las líneas de trabajo futuro que surgieron durante la implementación, las cuales permiten proyectar el prototipo hacia una solución más robusta, escalable y alineada con un escenario institucional real.

7.1. Conclusiones

El desarrollo de *BioAsist* permitió demostrar la factibilidad de implementar un control de asistencia basado en biometría, integrando hardware embebido, comunicación inalámbrica y un servidor web. El prototipo alcanzó a cumplir los objetivos principales definidos: el enrolamiento y validación de usuarios mediante huella dactilar, la transmisión de registros hacia una base de datos central y la visualización de estos registros a través de una interfaz web con perfiles diferenciados.

Las pruebas realizadas mostraron que el tiempo de registro es adecuado para cursos de tamaño medio, y que la tasa de fallos de reconocimiento se mantiene en un rango bajo, atribuible principalmente al posicionamiento del dedo en el *BioAsist-Device*. Asimismo, la comunicación con el servidor se mantuvo estable, validando la robustez de la arquitectura seleccionada.

Entre las principales conclusiones, destacan:

- El uso de huellas dactilares constituye una solución práctica y accesible para el registro de asistencia, evitando suplantaciones y reduciendo el tiempo destinado a esta tarea.
- La elección del microcontrolador ESP32 resultó adecuada, dado que permitió integrar el lector biométrico con conectividad Wi-Fi y mantener la simplicidad de desarrollo gracias a su compatibilidad con el entorno Arduino.
- La plataforma web de *BioAsist* entregó las funcionalidades básicas necesarias para la consulta y exportación de registros, aunque su estructura podría beneficiarse del uso de frameworks modernos.
- Si bien no se implementó la sincronización offline, la arquitectura del sistema permite proyectar mejoras que lo harían escalable a un escenario institucional real.

En conclusión, *BioAsist* cumplió con su propósito de validar una solución biométrica para el control de asistencia en entornos académicos. A partir de este prototipo, se abre un camino de desarrollo hacia una plataforma más robusta, escalable y segura, capaz de integrarse de manera efectiva a la infraestructura tecnológica de una universidad.

7.2. Trabajo Futuro

Si bien *BioAsist* cumple con los objetivos planteados, durante su implementación surgieron limitaciones y oportunidades de mejora que pueden guiar trabajos futuros para lograr una solución más completa, escalable y alineada con el entorno universitario real.

Sincronización y verificación en línea

El sistema actual opera con un mecanismo basado en *polling*, lo que impide mantener al *BioAsist-Device* en estado constante de captura de asistencia. Este enfoque es suficiente en una etapa de prototipo, pero limita la fluidez en escenarios reales. Como trabajo futuro, se identifican tres posibles mejoras:

- **Enrolamiento centralizado:** en lugar de almacenar huellas en cada *BioAsist-Device*, el proceso de enrolamiento podría realizarse de forma única en un servidor central. De esta manera, un estudiante no tendría que repetir el enrolamiento en cada sala o con cada docente, y las plantillas biométricas se gestionarían de forma unificada. Este enfoque ya es utilizado por soluciones comerciales de control de acceso, como las ofrecidas por ZKTeco y Suprema, donde un usuario enrolado en el sistema central queda automáticamente disponible en todos los terminales conectados.
- **Verificación en línea:** el *BioAsist-Device* podría capturar la huella, transmitirla vía HTTPS al servidor y recibir la validación en tiempo real. Esta opción asegura un control totalmente centralizado, pero depende fuertemente de la conectividad de red. En caso de fallas o latencia alta, la experiencia del usuario podría verse afectada.
- **Sincronización previa de plantillas:** una estrategia híbrida consiste en descargar, al inicio de cada clase, las plantillas de los alumnos inscritos en ese ramo desde el servidor al *BioAsist-Device*. Durante la clase, el dispositivo funciona en modo offline, registrando las asistencias localmente. Una vez finalizada la sesión o restablecida la conexión, los datos se sincronizan con el servidor central. Este modelo equilibra autonomía local y consistencia global.

El rediseño hacia alguna de estas alternativas permitiría mantener al *BioAsist-Device* siempre en estado de captura de asistencia, evitando depender de endpoints externos para habilitar el proceso. Además, abre la posibilidad de implementar un esquema de *store and forward*, donde las asistencias se almacenan en la memoria no volátil del microcontrolador (NVS en ESP32) y se sincronizan automáticamente al servidor tan pronto haya conectividad.

Integración con redes institucionales

Actualmente *BioAsist* opera sobre redes Wi-Fi estándar basadas en SSID y contraseña. Sin embargo, la Universidad Técnica Federico Santa María utiliza la red institucional **eduroam**, la cual se basa en el protocolo WPA2-Enterprise. Espressif provee bibliotecas para habilitar este tipo

de autenticación en dispositivos ESP32, pero por limitaciones de tiempo no se logró integrar ni validar.

Como trabajo futuro, la incorporación de soporte para eduroam resulta fundamental para un despliegue real, ya que permitiría conectar los *BioAsist-Device* directamente a la infraestructura de red universitaria, garantizando mayor seguridad y cobertura en todas las dependencias.

Soporte para múltiples lectores

En un escenario de adopción institucional, es probable que se requiera operar con múltiples *BioAsist-Device* distribuidos en diferentes salas de clases. Para ello, resulta necesario asignar un identificador único a cada dispositivo, de manera que los registros puedan ser diferenciados en la base de datos central.

Una alternativa práctica consiste en utilizar la dirección MAC del módulo ESP32 como identificador único. Otra opción es asignar manualmente un ID persistente almacenado en la memoria no volátil del microcontrolador, lo cual facilitaría la administración en escenarios con reemplazo de equipos o reconfiguraciones.

Mejoras en la plataforma web

La interfaz web actual de *BioAsist*, desarrollada en HTML, CSS, JavaScript y PHP, cumplió con los requerimientos funcionales planteados. No obstante, el uso de frameworks modernos permitiría mejorar aspectos de escalabilidad, mantenibilidad y experiencia de usuario.

Frameworks como Django (Python), Laravel (PHP), o soluciones frontend como React o Angular, ofrecen arquitecturas más robustas y facilitan la integración de características adicionales como:

- Visualización dinámica de estadísticas de asistencia.
- Notificaciones en tiempo real.
- Roles de usuario más complejos y gestión avanzada de permisos.

Otras mejoras relevantes

Además de los puntos anteriores, se identifican otras áreas donde el sistema podría ser perfeccionado:

- **Seguridad de datos biométricos:** en un entorno productivo se debería cifrar tanto la transmisión como el almacenamiento de plantillas biométricas, aplicando estándares de seguridad (ej. AES o TLS) y considerando las normativas de protección de datos personales.
- **Escalabilidad en la nube:** integrar la solución con plataformas cloud (como AWS IoT Core o Azure IoT Hub) permitiría administrar un número mayor de dispositivos y usuarios simultáneos sin comprometer el rendimiento.
- **Interoperabilidad:** habilitar integraciones automáticas con sistemas académicos existentes (ej. Moodle o Banner), de modo que la asistencia registrada se sincronice directamente con las plataformas docentes.
- **Analítica de datos:** extender la interfaz web con módulos de análisis que permitan identificar patrones de asistencia, ausentismo recurrente o correlaciones con el desempeño académico.

En síntesis, el trabajo futuro se enfoca en cuatro ejes principales: (i) rediseñar el sistema de sincronización y verificación, explorando alternativas de enrolamiento centralizado, verificación en línea o sincronización previa de plantillas; (ii) integrar *BioAsist* a la red institucional eduroam; (iii) habilitar soporte para múltiples *BioAsist-Device* distribuidos; y (iv) modernizar la plataforma web. Junto a mejoras adicionales en seguridad, escalabilidad y analítica, estas líneas de trabajo permitirán transformar el prototipo actual en una solución robusta, segura y escalable, apta para un despliegue institucional en la universidad.

Bibliografía

[1] Dorlo Technologies. (s.f.) Optical sensor vs. capacitive sensor. Consultado en noviembre de 2025. [Online]. Available: https://www.dorlotec.com/new/optical-sensor-vs-capacitive-sensor.html

[2] Hangzhou Grow Technology, *R307 Fingerprint Module User Manual*, 2011, v1.20. [Online]. Available: https://www.openhacks.com/uploadsproductos/r307_fingerprint_module_user_manual.pdf

[3] Espressif Systems, *ESP32 Technical Reference Manual*, 2022, disponible en línea. Último acceso: septiembre 2025. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf

[4] A. K. Jain, A. Ross, and S. Prabhakar, "An introduction to biometric recognition," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, no. 1, pp. 4–20, January 2004, accedido en septiembre 2025. [Online]. Available: https://ieeexplore.ieee.org/document/1262027

[5] "How do fingerprint scanners work? optical vs. capacitive," https://www.arrow.com/en/research-and-events/articles/how-fingerprint-sensors-work, accedido en septiembre 2025.

[6] "How fingerprint scanners work — optical, capacitive more," https://www.androidauthority.com/how-fingerprint-scanners-work-670934/, accedido en septiembre 2025.

[7] E. Rahkoyo, S. Yangdol, B. Kaur, D. Vaithiyanathan, and P. Verma, "Iot-based fingerprint attendance system: Enhancing efficiency and security in educational and organizational settings," in *2024 International Conference on Advances in Modern Age Technologies for Health and Engineering Science (AMATHE)*. Shivamogga, India: IEEE, May 2024, pp. 1–6, accedido en septiembre 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10582124

[8] SparkFun Electronics. (s.f.) Logic levels: Ttl vs cmos vs rs-232. Consultado en noviembre de 2025. [Online]. Available: https://learn.sparkfun.com/tutorials/logic-levels

[9] O. Standard, "Mqtt version 5.0," OASIS, Tech. Rep., 2019. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html

[10] I. Fette and A. Melnikov, "The websocket protocol," RFC 6455, 2011. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6455

[11] W3C, "Html 4.01 specification: application/x-www-form-urlencoded," 1999. [Online]. Available: https://www.w3.org/TR/html401/interact/forms.html#h-17.13.4

[12] Adafruit, "Adafruit fingerprint sensor library," 2025, disponible en GitHub. Último acceso: marzo 2025. [Online]. Available: https://github.com/adafruit/Adafruit-Fingerprint-Sensor-Library

[13] Arduino LLC, *Arduino IDE Documentation*, 2024, accessed: 2025-01-10. [Online]. Available: https://docs.arduino.cc/software/ide/

[14] PlatformIO Labs, *PlatformIO Documentation*, 2024, accessed: 2025-01-10. [Online]. Available: https://docs.platformio.org/

[15] Espressif Systems, *ESP-IDF Programming Guide*, 2024, accessed: 2025-01-10. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/

[16] "As608 fingerprint sensor datasheet," https://handsontec.com/dataspecs/sensor/AS608%20Finger%20print%20Sensor.pdf, accedido en septiembre 2025.

[17] "Gt-521f32 fingerprint scanner module datasheet," https://cdn.sparkfun.com/assets/learn_tutorials/7/2/3/GT-521FX2_datasheet_V1.1__003_.pdf, accedido en septiembre 2025.

[18] "Adafruit optical fingerprint sensor (id 751)," https://www.adafruit.com/product/751, accedido en septiembre 2025.

[19] "Adafruit capacitive fingerprint sensor (id 4651)," https://www.adafruit.com/product/4651, accedido en septiembre 2025.

[20] H. Ltd., *HD44780U: LCD Controller/Driver*, 1998, datasheet. [Online]. Available: https://www.sparkfun.com/datasheets/LCD/HD44780.pdf

[21] N. Semiconductors, *PCF8574: Remote 8-bit I/O Expander for I2C Bus*, 2015, datasheet. [Online]. Available: https://www.nxp.com/docs/en/data-sheet/PCF8574_PCF8574A.pdf

[22] F. de Brabander, "Arduino liquidcrystal i2c library," 2025, disponible en GitHub. Último acceso: marzo 2025. [Online]. Available: https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library

[23] Espressif Systems, "Arduino core for the esp32," 2025, disponible en GitHub. Último acceso: marzo 2025. [Online]. Available: https://github.com/espressif/arduino-esp32

[24] E. Systems, "Preferences (non-volatile storage) – arduino-esp32 api reference," 2025, documentación oficial. Último acceso: marzo 2025. [Online]. Available: https://docs.espressif.com/projects/arduino-esp32/en/latest/api/preferences.html

[25] Espressif Systems, "Esp32 wifi library (wifi.h) – official examples," https://github.com/espressif/arduino-esp32/tree/master/libraries/WiFi, 2023, ejemplos oficiales incluidos en el core Arduino para ESP32.

Apéndice A

Codigo fuente relevante

Este apéndice reúne extractos de código que respaldan la implementación descrita en el capítulo 5.3 (backend PHP) y capítulo 5.2 (firmware ESP32). Se incluyen únicamente las partes más relevantes para entender la lógica, omitiendo configuraciones triviales o repetitivas.

A.1. Firmware en ESP32

A.1.1. setup()

```
void setup() {
  Serial.begin(9600);  // opcional para debug

  lcd.init();
  lcd.backlight();
  lcdMsg("Init...", "WiFi+FP+NVS");
  delay(700);

  nvsBegin();
  if (loadNextFid() == 0) saveNextFid(1);

  wifiConnect();

  serialPort.begin(57600, SERIAL_8N1, FP_RX_PIN, FP_TX_PIN);
  finger.begin(57600);
  delay(5);

  if (!finger.verifyPassword()) {
    lcdMsg("Sensor huella", "NO detectado");
    while (1) delay(1);
  }
#ifdef ADMIN_WIPE_ON_BOOT
  lcdMsg("Modo admin", "Wipe total");
  wipeSensorAndLocal();
  while (1) { delay(1000); }  // seguridad
#endif
```

```
  lcdMsg("Sensor OK", "Leyendo params");
  finger.getParameters();
  lcdParam("Capacity", finger.capacity);
  lcdParam("Security", finger.security_level);
  finger.getTemplateCount();
  lcdParam("Templates", finger.templateCount);

  lcdMsg("Polling activo", "Enrol+Asistencia");
}
```

A.1.2. loop()

```
void loop() {
  static uint32_t lastPoll = 0;

  // Heartbeat visual
  lcdDotsTick();
  delay(120);

  // 1) Polling (cada POLL_INTERVAL_MS)
  if (millis() - lastPoll >= POLL_INTERVAL_MS) {
    lastPoll = millis();

    // 1.a) Enrolamiento pendiente
    uint16_t user_id = 0;
    if (pollForEnrollment(user_id)) {
      uint16_t fid = 0;
      bool ok = doEnrollmentAuto(user_id, fid);
      if (ok && wifiEnsure()) {
        // Confirmar al servidor (Procesar Enrolamiento)
        int code;
        String resp;
        const String url = BASE_URL + EP_CONFIRM_ENROL;
        const String body = "id_usuario=" + String(user_id) + "&finger_id=" + String(fid);
        bool sent = httpPostForm(url, body, code, resp);
        lcdMsg((sent && code == 200) ? "Srv Enrol OK" : "Srv Enrol FAIL",
               (sent ? ("code " + String(code)) : "HTTP err"));
        delay(900);
      }
      lcdMsg(ok ? "Enrolado OK" : "Enrolado fallo", "UID " + String(user_id));
      delay(1000);
      lcdMsg("Polling activo", "Enrol+Asistencia");
    }

    // 1.b) Estado de asistencia (activa / inactiva) -> lee id_ramo del endpoint
    bool activa = false;
    uint16_t ramoEncontrado = 0;
```

```
    if (pollAttendanceState(activa, ramoEncontrado)) {
      asistenciaActiva = activa;
      currentRamo = ramoEncontrado;  // 0 si no hay activo

      if (asistenciaActiva && currentRamo != 0) {
        lcdMsg("Asistencia ACTIVA", "Ramo " + String(currentRamo));
      } else {
        lcdMsg("Asistencia INACT", "");
      }
      delay(800);
    }
  }
```

A.2. Código de enrolamiento

A.2.1. pollForEnrollment()

```
bool pollForEnrollment(uint16_t &user_id) {
  user_id = 0;
  if (!wifiEnsure()) return false;

  int code;
  String resp;
  const String url = BASE_URL + EP_POLL_ENROL;

  bool ok = httpGet(url, code, resp);
  if (!(ok && code == 200)) return false;

  if (resp.indexOf("sin solicitudes") >= 0) return false;

  // Parsear id_usuario (sin ArduinoJson)
  int pos = resp.indexOf("\"id_usuario\"");
  if (pos < 0) pos = resp.indexOf("id_usuario");
  if (pos < 0) return false;

  int sep = resp.indexOf(':', pos);
  if (sep < 0) sep = resp.indexOf('=', pos);
  if (sep < 0) return false;

  int i = sep + 1;
  while (i < (int)resp.length() && (resp[i] == ' ' || resp[i] == '\"')) i++;
  String num = "";
  while (i < (int)resp.length() && isDigit((unsigned char)resp[i])) num += resp[i++];
  uint32_t val = num.toInt();
  if (val == 0 || val > 65535) return false;
```

```
  user_id = (uint16_t)val;
  return true;
}
```

A.2.2. doEnrollmentAuto()

```
bool doEnrollmentAuto(uint16_t user_id, uint16_t &out_fid) {
  out_fid = 0;

  finger.getTemplateCount();
  uint16_t maxId = (uint16_t)min((uint16_t)finger.capacity, (uint16_t)127);
  if (maxId < 1) maxId = 127;

  uint16_t start = loadNextFid();
  if (start < 1 || start > maxId) start = 1;

  int freeSlot = findNextFreeSlot((uint8_t)start, (uint8_t)maxId);
  if (freeSlot < 0) {
    lcdMsg("Sensor lleno", "");
    return false;
  }
  uint16_t fid = (uint16_t)freeSlot;

  int p = -1;

  // === NUEVO: control de timeout y re-chequeo ===
  const uint32_t ENROLL_MAX_MS = 60000; // 60s total
  const uint32_t RECHK_MS      = 1200;  // re-chequear server cada ~1.2s
  uint32_t tStart   = millis();
  uint32_t lastRechk= 0;

  // 1) Primer escaneo
  lcdMsg("Coloque el dedo", "FID " + String(fid));
  while (p != FINGERPRINT_OK) {
    p = finger.getImage();
    if (p == FINGERPRINT_OK) lcdMsg("Imagen tomada", "");
    else if (p == FINGERPRINT_NOFINGER) {
      lcdMsg("Esperando dedo", "");
      lcdDotsTick();
      delay(120);
    } else if (p == FINGERPRINT_PACKETRECIEVEERR) {
      lcdMsg("Error comm", "reintente");
      delay(600);
    } else if (p == FINGERPRINT_IMAGEFAIL) {
      lcdMsg("Error imagen", "reintente");
      delay(600);
    } else {
      lcdMsg("Error desconoc.", "");
```

```
    delay(600);
  }

  // chequeo de cancelación + timeout
  if (millis() - lastRechk > RECHK_MS) {
    lastRechk = millis();
    if (!isEnrollmentPending(user_id)) {
      lcdMsg("Enrolamiento", "cancelado");
      return false;
    }
  }
  if (millis() - tStart > ENROLL_MAX_MS) {
    lcdMsg("Enrolamiento", "timeout");
    return false;
  }
}

p = finger.image2Tz(1);
if (p != FINGERPRINT_OK) {
  switch (p) {
    case FINGERPRINT_IMAGEMESS: lcdMsg("Imagen sucia", "reintente"); break;
    case FINGERPRINT_PACKETRECIEVEERR: lcdMsg("Error comm", ""); break;
    case FINGERPRINT_FEATUREFAIL:
    case FINGERPRINT_INVALIDIMAGE: lcdMsg("Sin rasgos", "reintente"); break;
    default: lcdMsg("Error desconoc.", ""); break;
  }
  return false;
}
lcdMsg("Convertido 1/2", "");

// 2) Retirar dedo
lcdMsg("Retire el dedo", "");
delay(1000);
while (true) {
  int g = finger.getImage();
  if (g == FINGERPRINT_NOFINGER) break;
  delay(50);

  // chequeo de cancelación + timeout
  if (millis() - lastRechk > RECHK_MS) {
    lastRechk = millis();
    if (!isEnrollmentPending(user_id)) {
      lcdMsg("Enrolamiento", "cancelado");
      return false;
    }
  }
  if (millis() - tStart > ENROLL_MAX_MS) {
    lcdMsg("Enrolamiento", "timeout");
```

```
    return false;
  }
}

// 3) Segundo escaneo
lcdMsg("Mismo dedo", "nuevamente");
p = -1;
while (p != FINGERPRINT_OK) {
  p = finger.getImage();
  if (p == FINGERPRINT_OK) lcdMsg("Imagen tomada", "");
  else if (p == FINGERPRINT_NOFINGER) {
    lcdMsg("Esperando dedo", "");
    lcdDotsTick();
    delay(120);
  } else if (p == FINGERPRINT_PACKETRECIEVEERR) {
    lcdMsg("Error comm", "");
    delay(600);
  } else if (p == FINGERPRINT_IMAGEFAIL) {
    lcdMsg("Error imagen", "");
    delay(600);
  } else {
    lcdMsg("Error desconoc.", "");
    delay(600);
  }

  // chequeo de cancelación + timeout
  if (millis() - lastRechk > RECHK_MS) {
    lastRechk = millis();
    if (!isEnrollmentPending(user_id)) {
      lcdMsg("Enrolamiento", "cancelado");
      return false;
    }
  }
  if (millis() - tStart > ENROLL_MAX_MS) {
    lcdMsg("Enrolamiento", "timeout");
    return false;
  }
}

p = finger.image2Tz(2);
if (p != FINGERPRINT_OK) {
  switch (p) {
    case FINGERPRINT_IMAGEMESS: lcdMsg("Imagen sucia", "reintente"); break;
    case FINGERPRINT_PACKETRECIEVEERR: lcdMsg("Error comm", ""); break;
    case FINGERPRINT_FEATUREFAIL:
    case FINGERPRINT_INVALIDIMAGE: lcdMsg("Sin rasgos", "reintente"); break;
    default: lcdMsg("Error desconoc.", ""); break;
  }
```

```
    return false;
  }
  lcdMsg("Convertido 2/2", "");

  // 4) Crear modelo
  lcdMsg("Creando modelo", "FID " + String(fid));
  p = finger.createModel();
  if (p != FINGERPRINT_OK) {
    if (p == FINGERPRINT_PACKETRECIEVEERR) lcdMsg("Error comm", "");
    else if (p == FINGERPRINT_ENROLLMISMATCH) lcdMsg("No coinciden", "reintente");
    else lcdMsg("Error desconoc.", "");
    return false;
  }
  lcdMsg("Coincidencia OK", "");

  // 5) Guardar en el slot
  lcdMsg("Guardando...", "FID " + String(fid));
  p = finger.storeModel(fid);
  if (p != FINGERPRINT_OK) {
    lcdMsg("Error guardando", "");
    return false;
  }

  // 6) Persistir local y preparar próximo
  saveFingerUserMap(fid, user_id);
  uint16_t next = (fid % maxId) + 1;
  saveNextFid(next);

  out_fid = fid;
  return true;
}
```

A.2.3. isEnrollmentPending()

```
bool isEnrollmentPending(uint16_t user_id) {
  if (!wifiEnsure()) return true;
  int code; String resp;
  String url = BASE_URL + EP_ESTADO_ENROL + "?id_usuario=" + String(user_id);
  bool ok = httpGet(url, code, resp);
  if (!(ok && code == 200)) return true;

  String s = resp; s.toLowerCase();
  if (s.indexOf("cancelado") >= 0) return false;
  if (s.indexOf("huella registrada") >= 0) return false;
  if (s.indexOf("pendiente") >= 0) return true;
  return true;
}
```

A.3. Código de asistencia

A.3.1. pollAttendanceState()

```
bool pollAttendanceState(bool &activa, uint16_t &ramoIdOut) {
  activa = false;
  ramoIdOut = 0;

  if (!wifiEnsure()) return false;

  int code;
  String resp;
  const String url = BASE_URL + EP_ESTADO_ASIS;  // SIN ?id_ramo
  if (!httpGet(url, code, resp) || code != 200) return false;

  String r = resp;
  r.toLowerCase();

  // activo
  int p = r.indexOf("\"activo\"");
  if (p < 0) p = r.indexOf("activo");
  if (p >= 0) {
    int sep = r.indexOf(':', p);
    if (sep > 0) {
      String tail = r.substring(sep + 1);
      activa = (tail.indexOf("true") >= 0 || tail.indexOf("1") >= 0);
    }
  }

  // id_ramo (solo si activo)
  if (activa) {
    int q = r.indexOf("\"id_ramo\"");
    if (q < 0) q = r.indexOf("id_ramo");
    if (q >= 0) {
      int sep = r.indexOf(':', q);
      if (sep > 0) {
        int i = sep + 1;
        while (i < (int)r.length() && !isDigit(r[i])) i++;
        String num = "";
        while (i < (int)r.length() && isDigit(r[i])) num += r[i++];
        if (num.length()) ramoIdOut = (uint16_t)num.toInt();
      }
    }
  }

  return true;
}
```

A.3.2. scanAndRegisterAttendance()

```
bool scanAndRegisterAttendance(uint16_t ramoId) {
  if (ramoId == 0) return false;

  // 1) Captura rápida
  int p = finger.getImage();
  if (p == FINGERPRINT_NOFINGER) {
    if (++noFingerDots % 8 == 0) lcdMsg("Asistencia ACTIVA", "Ramo " + String(ramoId));
    return false;
  }
  if (p == FINGERPRINT_PACKETRECIEVEERR) {
    lcdMsg("Err comm", "");
    delay(400);
    return false;
  }
  if (p == FINGERPRINT_IMAGEFAIL) {
    lcdMsg("Err imagen", "");
    delay(400);
    return false;
  }
  if (p != FINGERPRINT_OK) { return false; }

  lcdMsg("Imagen tomada", "");

  // 2) Convertir y buscar
  p = finger.image2Tz();
  if (p != FINGERPRINT_OK) {
    lcdMsg((p == FINGERPRINT_IMAGEMESS) ? "Imagen sucia" : (p ==
    FINGERPRINT_PACKETRECIEVEERR) ? "Err comm" : "Sin rasgos", "");
    delay(500);
    return false;
  }

  p = finger.fingerSearch();
  if (p != FINGERPRINT_OK) {
    lcdMsg((p == FINGERPRINT_NOTFOUND) ? "No coincide" : (p ==
    FINGERPRINT_PACKETRECIEVEERR) ? "Err comm" : "Err busqueda", "");
    delay(700);
    return false;
  }

  uint16_t fid = finger.fingerID;
  uint16_t conf = finger.confidence;

  // 3) Resolver id_usuario por mapeo local
  uint16_t user_id = getUserByFinger(fid);
  if (user_id == 0) {
```

```
    lcdMsg("FID " + String(fid), "Sin mapeo local");
    delay(900);
    return false;
  }

  // 4) Anti-doble registro
  uint32_t now = millis();
  auto it = lastPunchMs.find(user_id);
  if (it != lastPunchMs.end()) {
    if (now - it->second < PUNCH_COOLDOWN_MS) {
      lcdMsg("Ya registrado", "espere...");
      delay(900);
      return false;
    }
  }

  // 5) Registrar asistencia en servidor
  if (!wifiEnsure()) {
    lcdMsg("Sin WiFi", "No se registra");
    delay(800);
    return false;
  }

  int code;
  String resp;
  String url = BASE_URL + EP_REG_ASIS;
  String body = "id_usuario=" + String(user_id) + "&id_ramo=" + String(ramoId);
  bool sent = httpPostForm(url, body, code, resp);

  if (sent && code == 200) {
    lastPunchMs[user_id] = now;
    lcdMsg("Asistencia OK", "UID " + String(user_id));
    delay(1200);
    return true;
  } else {
    lcdMsg("Asist. FAIL", "code " + String(code));
    delay(900);
    return false;
  }
}
```

A.4. Persistencia en memoria NVS

A.4.1. saveFingerUserMap()

```
void saveFingerUserMap(uint16_t finger_id, uint16_t user_id) {
  nvsBegin();
```

```
  String key = "f_" + String(finger_id);
  prefs.putUShort(key.c_str(), user_id);
}
```

A.4.2. getUserByFinger()

```
uint16_t getUserByFinger(uint16_t finger_id) {
  nvsBegin();
  String key = "f_" + String(finger_id);
  return prefs.getUShort(key.c_str(), 0);
}
```

A.4.3. saveNextFid() y loadNextFid()

```
void saveNextFid(uint16_t fid) {
  nvsBegin();
  prefs.putUShort(KEY_NEXTFID, fid);
}

uint16_t loadNextFid() {
  nvsBegin();
  return prefs.getUShort(KEY_NEXTFID, 1);  // default 1
}
```

A.5. Funciones de red

A.5.1. wifiConnect() y wifiEnsure()

```
void wifiConnect() {
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  lcdMsg("WiFi conectando", WIFI_SSID);
  uint32_t t0 = millis();
  while (WiFi.status() != WL_CONNECTED && millis() - t0 < 15000) {
    lcdDotsTick();
    delay(250);
  }
  if (WiFi.status() == WL_CONNECTED) lcdMsg("WiFi OK", WiFi.localIP().toString());
  else lcdMsg("WiFi FAIL", "");
  delay(900);
}

bool wifiEnsure() {
  if (WiFi.status() == WL_CONNECTED) return true;
  wifiConnect();
  return WiFi.status() == WL_CONNECTED;
}
```

A.5.2. httpGet() y httpPostForm()

```
bool httpGet(const String &url, int &code, String &resp) {
  HTTPClient http;
  http.begin(url);
  http.setReuse(false);
  http.setTimeout(7000);
  http.addHeader("Cache-Control", "no-cache");
  http.addHeader("Pragma", "no-cache");
  code = http.GET();
  resp = http.getString();
  http.end();
  return (code > 0);
}

bool httpPostForm(const String &url, const String &body, int &code, String &resp) {
  HTTPClient http;
  http.begin(url);
  http.setReuse(false);
  http.setTimeout(7000);
  http.addHeader("Cache-Control", "no-cache");
  http.addHeader("Pragma", "no-cache");
  http.addHeader("Content-Type", "application/x-www-form-urlencoded");
  code = http.POST(body);
  resp = http.getString();
  http.end();
  return (code > 0);
}
```

A.6. Funciones de mantenimiento

A.6.1. wipeSensorAndLocal()

```
bool wipeSensorAndLocal() {
  lcdMsg("Borrando sensor", "huellas...");
  uint8_t r = finger.emptyDatabase();
  if (r != FINGERPRINT_OK) {
    lcdMsg("Error borrando", "code " + String(r));
    return false;
  }
  lcdMsg("Sensor limpio", "");
  delay(600);

  // Limpia el mapeo y reinicia contador local
  clearLocalFingerprintMap();  // o usa clearLocalNamespaceFast();
  lcdMsg("NVS reiniciada", "next_fid=1");
  delay(800);
  return true;
```

```
}
```

A.7. Backend PHP

A.7.1. verificar_enrolamiento.php

```
<?php
header('Content-Type: application/json');
$conn = new mysqli("localhost", "root", "", "asistencia");

$result = $conn->query("SELECT id, id_usuario FROM enrolamientos WHERE
                        estado = 'pendiente' LIMIT 1");

if ($row = $result->fetch_assoc()) {
    echo json_encode([
        'id_enrolamiento' => $row['id'],
        'id_usuario' => $row['id_usuario']
    ]);
} else {
    echo json_encode(['mensaje' => 'sin solicitudes']);
}
?>
```

A.7.2. procesar_enrolamiento.php

```
<?php
header('Content-Type: application/json');

$conn = new mysqli("localhost", "root", "", "asistencia");
if ($conn->connect_error) {
    die(json_encode(["error" => "Error de conexión a la base de datos."]));
}

$id_usuario = $_POST['id_usuario'] ?? null;
$finger_id = $_POST['finger_id'] ?? null;

if (!$id_usuario || !$finger_id) {
    http_response_code(400);
    echo json_encode(["error" => "Faltan parámetros"]);
    exit();
}

// 1. Verificar que exista un enrolamiento pendiente
$stmt = $conn->prepare("SELECT id FROM enrolamientos WHERE id_usuario = ?
                    AND estado = 'pendiente' ORDER BY fecha DESC LIMIT 1");
$stmt->bind_param("i", $id_usuario);
$stmt->execute();
$result = $stmt->get_result();
```

```
if ($row = $result->fetch_assoc()) {
    $id_enrolamiento = $row['id'];

    // 2. Actualizar usuarios con el nuevo finger_id
    $updateUser = $conn->prepare("UPDATE usuarios SET finger_id = ? WHERE id = ?");
    $updateUser->bind_param("ii", $finger_id, $id_usuario);
    $updateUser->execute();

    // 3. Actualizar estado del enrolamiento
    $estado = " Huella registrada";
    $updateEnroll = $conn->prepare("UPDATE enrolamientos SET estado = ? WHERE id = ?");
    $updateEnroll->bind_param("si", $estado, $id_enrolamiento);
    $updateEnroll->execute();

    echo json_encode(["success" => "Huella registrada para usuario $id_usuario
                                    con ID $finger_id"]);
} else {
    echo json_encode(["error" => "No hay solicitud de enrolamiento
                        pendiente para este usuario."]);
}
?>
```

A.7.3. verificar_estado_enrolamiento.php

```
<?php
ini_set('display_errors', 1);
ini_set('display_startup_errors', 1);
error_reporting(E_ALL);

header('Content-Type: application/json');

$conn = new mysqli("localhost", "root", "", "asistencia");

$id_usuario = $_GET['id_usuario'] ?? null;
if (!$id_usuario) {
    echo json_encode(['estado' => 'error', 'msg' => 'Sin ID']);
    exit;
}

$stmt = $conn->prepare("
    SELECT estado
    FROM enrolamientos
    WHERE id_usuario = ?
    ORDER BY fecha DESC
    LIMIT 1
");
$stmt->bind_param("i", $id_usuario);
```

```
$stmt->execute();
$result = $stmt->get_result();

if ($row = $result->fetch_assoc()) {
    echo json_encode([
        'estado' => $row['estado'],
    ]);
} else {
    echo json_encode(['estado' => 'sin registro', 'finger_id' => null]);
}
?>
```

A.7.4. estado_actual_asistencia.php

```
<?php
// estado_actual_asistencia.php
header('Content-Type: application/json; charset=utf-8');

mysqli_report(MYSQLI_REPORT_OFF);
$conn = new mysqli("localhost", "root", "", "asistencia");
if ($conn->connect_error) {
    http_response_code(500);
    echo json_encode(["activo" => false, "id_ramo" => null, "error" => "DB_CONN"]);
    exit();
}

// Busca un ramo ACTIVO para HOY
$stmt = $conn->prepare("
    SELECT id_ramo
    FROM estado_asistencia
    WHERE fecha = CURDATE() AND activo = 1
    ORDER BY id DESC
    LIMIT 1
");
$stmt->execute();
$res = $stmt->get_result();
$row = $res ? $res->fetch_assoc() : null;

if ($row && isset($row['id_ramo'])) {
    echo json_encode(["activo" => true, "id_ramo" => (int)$row['id_ramo']]);
} else {
    echo json_encode(["activo" => false, "id_ramo" => null]);
}
?>
```

A.7.5. registrar_asistencia.php

```
<?php
```

```
// registrar_asistencia.php
header("Content-Type: application/json");
error_reporting(E_ALL);
ini_set('display_errors', 1);

$conn = new mysqli("localhost", "root", "", "asistencia");
if ($conn->connect_error) {
    http_response_code(500);
    echo json_encode(["ok" => false, "error" => "DB_CONN"]);
    exit();
}

// Parámetros esperados del dispositivo
$id_usuario = isset($_POST['id_usuario']) ? (int)$_POST['id_usuario'] : 0; // alumno
$id_ramo    = isset($_POST['id_ramo'])    ? (int)$_POST['id_ramo']    : 0;
$fecha      = date('Y-m-d');

if ($id_usuario <= 0 || $id_ramo <= 0) {
    http_response_code(400);
    echo json_encode(["ok" => false, "error" => "BAD_PARAMS"]);
    exit();
}

// 1) Verificar que el alumno exista y sea de tipo 'alumno'
$qa = $conn->prepare("SELECT id FROM usuarios WHERE id = ? AND tipo = 'alumno' LIMIT 1");
$qa->bind_param("i", $id_usuario);
$qa->execute();
$al = $qa->get_result()->fetch_assoc();
if (!$al) {
    http_response_code(404);
    echo json_encode(["ok" => false, "error" => "ALUMNO_NO_ENCONTRADO"]);
    exit();
}

// 2) Verificar que el alumno esté inscrito en el ramo
$qir = $conn->prepare("
    SELECT 1
    FROM alumnos_ramos
    WHERE id_alumno = ? AND id_ramo = ?
    LIMIT 1
");
$qir->bind_param("ii", $id_usuario, $id_ramo);
$qir->execute();
$inscrito = $qir->get_result()->fetch_row();
if (!$inscrito) {
    http_response_code(403);
    echo json_encode(["ok" => false, "error" => "ALUMNO_NO_INSCRITO"]);
    exit();
```

```
}

// 3) Verificar que la asistencia del ramo esté ACTIVA
$qe = $conn->prepare("
    SELECT activo
    FROM estado_asistencia
    WHERE id_ramo = ?
    ORDER BY id DESC
    LIMIT 1
");
$qe->bind_param("i", $id_ramo);
$qe->execute();
$er = $qe->get_result()->fetch_assoc();
if (!$er || (int)$er['activo'] !== 1) {
    http_response_code(403);
    echo json_encode(["ok" => false, "error" => "ASISTENCIA_INACTIVA"]);
    exit();
}

// 4) Registrar (o actualizar) asistencia
// Nota: se recomienda UNIQUE KEY (id_alumno, id_ramo, fecha) en la tabla 'asistencia'
$stmt = $conn->prepare("
    INSERT INTO asistencia (id_alumno, id_ramo, fecha, estado)
    VALUES (?, ?, ?, 'Presente')
    ON DUPLICATE KEY UPDATE estado = 'Presente'
");
$stmt->bind_param("iis", $id_usuario, $id_ramo, $fecha);

if ($stmt->execute()) {
    echo json_encode([
        "ok"        => true,
        "msg"       => "ASISTENCIA_REGISTRADA",
        "id_alumno" => $id_usuario,
        "id_ramo"   => $id_ramo,
        "fecha"     => $fecha
    ]);
} else {
    http_response_code(500);
    echo json_encode(["ok" => false, "error" => "INSERT_FAILED"]);
}
?>
```

A.7.6. activar_asistencia.php

y finalizar_asistencia.php

```
<?php
// activar_asistencia.php
```

```
mysqli_report(MYSQLI_REPORT_OFF);
$conn = new mysqli("localhost", "root", "", "asistencia");
if ($conn->connect_error) {
    http_response_code(500);
    die("Error de conexión: " . $conn->connect_error);
}

$id_ramo = isset($_POST['id_ramo']) ? (int)$_POST['id_ramo'] : 0;
if ($id_ramo <= 0) {
    http_response_code(400);
    echo " Parámetros inválidos.";
    exit();
}

$stmt = $conn->prepare("
    INSERT INTO estado_asistencia (id_ramo, activo, fecha, hora_inicio, hora_fin)
    VALUES (?, 1, CURDATE(), CURTIME(), NULL)
    ON DUPLICATE KEY UPDATE
        activo = 1,
        fecha = CURDATE(),
        hora_inicio = CURTIME(),
        hora_fin = NULL
");
$stmt->bind_param("i", $id_ramo);
$stmt->execute();
$stmt->close();

header("Location: pasar_asistencia.php?id_ramo={$id_ramo}&iniciado=1");
exit();


<?php
// desactivar_asistencia.php

mysqli_report(MYSQLI_REPORT_OFF);
$conn = new mysqli("localhost", "root", "", "asistencia");
if ($conn->connect_error) {
    http_response_code(500);
    die("Error de conexión: " . $conn->connect_error);
}

$id_ramo = isset($_POST['id_ramo']) ? (int)$_POST['id_ramo'] : 0;
if ($id_ramo <= 0) {
    http_response_code(400);
    echo " Parámetros inválidos.";
    exit();
}
```

```
$upd = $conn->prepare("
    UPDATE estado_asistencia
    SET activo = 0, hora_fin = CURTIME()
    WHERE id_ramo = ? AND fecha = CURDATE()
");
$upd->bind_param("i", $id_ramo);
$upd->execute();
$affected = $upd->affected_rows;
$upd->close();

$sel = $conn->prepare("
    SELECT ar.id_alumno
    FROM alumnos_ramos ar
    LEFT JOIN asistencia a
      ON a.id_alumno = ar.id_alumno
     AND a.id_ramo   = ar.id_ramo
     AND a.fecha     = CURDATE()
    WHERE ar.id_ramo = ? AND a.id IS NULL
");
$sel->bind_param("i", $id_ramo);
$sel->execute();
$res = $sel->get_result();

$ins = $conn->prepare("
    INSERT INTO asistencia (id_alumno, id_ramo, fecha, estado)
    VALUES (?, ?, CURDATE(), 'Ausente')
");

while ($row = $res->fetch_assoc()) {
    $id_alumno = (int)$row['id_alumno'];
    $ins->bind_param("ii", $id_alumno, $id_ramo);
    $ins->execute();
}
$ins->close();
$sel->close();

header("Location: pasar_asistencia.php?id_ramo={$id_ramo}&finalizado=1");
exit();
```

A.8. Pruebas Funcionales por modulo

A.8.1. Lector Biometrico R307 - Enrolar

```
    /***************************************************
  This is an example sketch for our optical Fingerprint sensor
```

```
  Designed specifically to work with the Adafruit BMP085 Breakout
  ----> http://www.adafruit.com/products/751

  These displays use TTL Serial to communicate, 2 pins are required to
  interface
  Adafruit invests time and resources providing this open source code,
  please support Adafruit and open-source hardware by purchasing
  products from Adafruit!

  Written by Limor Fried/Ladyada for Adafruit Industries.
  Small bug-fix by Michael cochez

  BSD license, all text above must be included in any redistribution
 ****************************************************/

#include <Adafruit_Fingerprint.h>


#if (defined(__AVR__) || defined(ESP8266)) && !defined(__AVR_ATmega2560__)
// For UNO and others without hardware serial, we must use software serial...
// pin #2 is IN from sensor (GREEN wire)
// pin #3 is OUT from arduino  (WHITE wire)
// Set up the serial port to use softwareserial..
SoftwareSerial mySerial(2, 3);

#else
// On Leonardo/M0/etc, others with hardware serial, use hardware serial!
// #0 is green wire, #1 is white
#define mySerial Serial2

#endif


Adafruit_Fingerprint finger = Adafruit_Fingerprint(&mySerial);

uint8_t id;

void setup()
{
  Serial.begin(9600);
  while (!Serial);  // For Yun/Leo/Micro/Zero/...
  delay(100);
  Serial.println("\n\nAdafruit Fingerprint sensor enrollment");

  // set the data rate for the sensor serial port
  finger.begin(57600);
```

```
  if (finger.verifyPassword()) {
    Serial.println("Found fingerprint sensor!");
  } else {
    Serial.println("Did not find fingerprint sensor :(");
    while (1) { delay(1); }
  }

  Serial.println(F("Reading sensor parameters"));
  finger.getParameters();
  Serial.print(F("Status: 0x")); Serial.println(finger.status_reg, HEX);
  Serial.print(F("Sys ID: 0x")); Serial.println(finger.system_id, HEX);
  Serial.print(F("Capacity: ")); Serial.println(finger.capacity);
  Serial.print(F("Security level: ")); Serial.println(finger.security_level);
  Serial.print(F("Device address: ")); Serial.println(finger.device_addr, HEX);
  Serial.print(F("Packet len: ")); Serial.println(finger.packet_len);
  Serial.print(F("Baud rate: ")); Serial.println(finger.baud_rate);
}

uint8_t readnumber(void) {
  uint8_t num = 0;

  while (num == 0) {
    while (! Serial.available());
    num = Serial.parseInt();
  }
  return num;
}

void loop()                     // run over and over again
{
  Serial.println("Ready to enroll a fingerprint!");
  Serial.println("Please type in the ID # (from 1 to 127) you want to save
                    this finger as...");
  id = readnumber();
  if (id == 0) {// ID #0 not allowed, try again!
     return;
  }
  Serial.print("Enrolling ID #");
  Serial.println(id);

  while (! getFingerprintEnroll() );
}

uint8_t getFingerprintEnroll() {

  int p = -1;
  Serial.print("Waiting for valid finger to enroll as #"); Serial.println(id);
  while (p != FINGERPRINT_OK) {
```

```
  p = finger.getImage();
  switch (p) {
  case FINGERPRINT_OK:
    Serial.println("Image taken");
    break;
  case FINGERPRINT_NOFINGER:
    Serial.print(".");
    break;
  case FINGERPRINT_PACKETRECIEVEERR:
    Serial.println("Communication error");
    break;
  case FINGERPRINT_IMAGEFAIL:
    Serial.println("Imaging error");
    break;
  default:
    Serial.println("Unknown error");
    break;
  }
}

// OK success!

p = finger.image2Tz(1);
switch (p) {
  case FINGERPRINT_OK:
    Serial.println("Image converted");
    break;
  case FINGERPRINT_IMAGEMESS:
    Serial.println("Image too messy");
    return p;
  case FINGERPRINT_PACKETRECIEVEERR:
    Serial.println("Communication error");
    return p;
  case FINGERPRINT_FEATUREFAIL:
    Serial.println("Could not find fingerprint features");
    return p;
  case FINGERPRINT_INVALIDIMAGE:
    Serial.println("Could not find fingerprint features");
    return p;
  default:
    Serial.println("Unknown error");
    return p;
}

Serial.println("Remove finger");
delay(2000);
p = 0;
while (p != FINGERPRINT_NOFINGER) {
```

```
  p = finger.getImage();
}
Serial.print("ID "); Serial.println(id);
p = -1;
Serial.println("Place same finger again");
while (p != FINGERPRINT_OK) {
  p = finger.getImage();
  switch (p) {
  case FINGERPRINT_OK:
    Serial.println("Image taken");
    break;
  case FINGERPRINT_NOFINGER:
    Serial.print(".");
    break;
  case FINGERPRINT_PACKETRECIEVEERR:
    Serial.println("Communication error");
    break;
  case FINGERPRINT_IMAGEFAIL:
    Serial.println("Imaging error");
    break;
  default:
    Serial.println("Unknown error");
    break;
  }
}

// OK success!

p = finger.image2Tz(2);
switch (p) {
  case FINGERPRINT_OK:
    Serial.println("Image converted");
    break;
  case FINGERPRINT_IMAGEMESS:
    Serial.println("Image too messy");
    return p;
  case FINGERPRINT_PACKETRECIEVEERR:
    Serial.println("Communication error");
    return p;
  case FINGERPRINT_FEATUREFAIL:
    Serial.println("Could not find fingerprint features");
    return p;
  case FINGERPRINT_INVALIDIMAGE:
    Serial.println("Could not find fingerprint features");
    return p;
  default:
    Serial.println("Unknown error");
    return p;
```

```
  }

  // OK converted!
  Serial.print("Creating model for #");  Serial.println(id);

  p = finger.createModel();
  if (p == FINGERPRINT_OK) {
    Serial.println("Prints matched!");
  } else if (p == FINGERPRINT_PACKETRECIEVEERR) {
    Serial.println("Communication error");
    return p;
  } else if (p == FINGERPRINT_ENROLLMISMATCH) {
    Serial.println("Fingerprints did not match");
    return p;
  } else {
    Serial.println("Unknown error");
    return p;
  }

  Serial.print("ID "); Serial.println(id);
  p = finger.storeModel(id);
  if (p == FINGERPRINT_OK) {
    Serial.println("Stored!");
  } else if (p == FINGERPRINT_PACKETRECIEVEERR) {
    Serial.println("Communication error");
    return p;
  } else if (p == FINGERPRINT_BADLOCATION) {
    Serial.println("Could not store in that location");
    return p;
  } else if (p == FINGERPRINT_FLASHERR) {
    Serial.println("Error writing to flash");
    return p;
  } else {
    Serial.println("Unknown error");
    return p;
  }

  return true;
}
```

A.8.2. Lector Biometrico R307 - Registrar

```
    /**************************************************
  This is an example sketch for our optical Fingerprint sensor

  Designed specifically to work with the Adafruit BMP085 Breakout
  ----> http://www.adafruit.com/products/751
```

```
  These displays use TTL Serial to communicate, 2 pins are required to
  interface
  Adafruit invests time and resources providing this open source code,
  please support Adafruit and open-source hardware by purchasing
  products from Adafruit!

  Written by Limor Fried/Ladyada for Adafruit Industries.
  BSD license, all text above must be included in any redistribution
 ****************************************************/


#include <Adafruit_Fingerprint.h>


#if (defined(__AVR__) || defined(ESP8266)) && !defined(__AVR_ATmega2560__)
// For UNO and others without hardware serial, we must use software serial...
// pin #2 is IN from sensor (GREEN wire)
// pin #3 is OUT from arduino  (WHITE wire)
// Set up the serial port to use softwareserial..
SoftwareSerial mySerial(2, 3);

#else
// On Leonardo/M0/etc, others with hardware serial, use hardware serial!
// #0 is green wire, #1 is white
#define mySerial Serial1

#endif


Adafruit_Fingerprint finger = Adafruit_Fingerprint(&mySerial);

void setup()
{
  Serial.begin(9600);
  while (!Serial);  // For Yun/Leo/Micro/Zero/...
  delay(100);
  Serial.println("\n\nAdafruit finger detect test");

  // set the data rate for the sensor serial port
  finger.begin(57600);
  delay(5);
  if (finger.verifyPassword()) {
    Serial.println("Found fingerprint sensor!");
  } else {
    Serial.println("Did not find fingerprint sensor :(");
    while (1) { delay(1); }
  }
```

```
  Serial.println(F("Reading sensor parameters"));
  finger.getParameters();
  Serial.print(F("Status: 0x")); Serial.println(finger.status_reg, HEX);
  Serial.print(F("Sys ID: 0x")); Serial.println(finger.system_id, HEX);
  Serial.print(F("Capacity: ")); Serial.println(finger.capacity);
  Serial.print(F("Security level: ")); Serial.println(finger.security_level);
  Serial.print(F("Device address: ")); Serial.println(finger.device_addr, HEX);
  Serial.print(F("Packet len: ")); Serial.println(finger.packet_len);
  Serial.print(F("Baud rate: ")); Serial.println(finger.baud_rate);

  finger.getTemplateCount();

  if (finger.templateCount == 0) {
    Serial.print("Sensor doesn't contain any fingerprint data. Please run the
                  'enroll' example.");
  }
  else {
    Serial.println("Waiting for valid finger...");
    Serial.print("Sensor contains ");
    Serial.print(finger.templateCount);
    Serial.println(" templates");
  }
}

void loop()                     // run over and over again
{
  getFingerprintID();
  delay(50);            //don't ned to run this at full speed.
}

uint8_t getFingerprintID() {
  uint8_t p = finger.getImage();
  switch (p) {
    case FINGERPRINT_OK:
      Serial.println("Image taken");
      break;
    case FINGERPRINT_NOFINGER:
      Serial.println("No finger detected");
      return p;
    case FINGERPRINT_PACKETRECIEVEERR:
      Serial.println("Communication error");
      return p;
    case FINGERPRINT_IMAGEFAIL:
      Serial.println("Imaging error");
      return p;
    default:
      Serial.println("Unknown error");
```

```
    return p;
  }

  // OK success!

  p = finger.image2Tz();
  switch (p) {
    case FINGERPRINT_OK:
      Serial.println("Image converted");
      break;
    case FINGERPRINT_IMAGEMESS:
      Serial.println("Image too messy");
      return p;
    case FINGERPRINT_PACKETRECIEVEERR:
      Serial.println("Communication error");
      return p;
    case FINGERPRINT_FEATUREFAIL:
      Serial.println("Could not find fingerprint features");
      return p;
    case FINGERPRINT_INVALIDIMAGE:
      Serial.println("Could not find fingerprint features");
      return p;
    default:
      Serial.println("Unknown error");
      return p;
  }

  // OK converted!
  p = finger.fingerSearch();
  if (p == FINGERPRINT_OK) {
    Serial.println("Found a print match!");
  } else if (p == FINGERPRINT_PACKETRECIEVEERR) {
    Serial.println("Communication error");
    return p;
  } else if (p == FINGERPRINT_NOTFOUND) {
    Serial.println("Did not find a match");
    return p;
  } else {
    Serial.println("Unknown error");
    return p;
  }

  // found a match!
  Serial.print("Found ID #"); Serial.print(finger.fingerID);
  Serial.print(" with confidence of "); Serial.println(finger.confidence);

  return finger.fingerID;
}
```

```
// returns -1 if failed, otherwise returns ID #
int getFingerprintIDez() {
  uint8_t p = finger.getImage();
  if (p != FINGERPRINT_OK)  return -1;

  p = finger.image2Tz();
  if (p != FINGERPRINT_OK)  return -1;

  p = finger.fingerFastSearch();
  if (p != FINGERPRINT_OK)  return -1;

  // found a match!
  Serial.print("Found ID #"); Serial.print(finger.fingerID);
  Serial.print(" with confidence of "); Serial.println(finger.confidence);
  return finger.fingerID;
}
```

A.8.3. Prueba LCD I2C 16x2

```
    //YWROBOT
//Compatible with the Arduino IDE 1.0
//Library version:1.1
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27,16,2);  // set the LCD address to 0x27 for a 16
                                   //chars and 2 line display

void setup()
{
  lcd.init();                      // initialize the lcd
  // Print a message to the LCD.
  lcd.backlight();
  lcd.setCursor(0,0);
  lcd.print("Prueba ELO308");
  lcd.setCursor(0,1);
  lcd.print("Display 16x2");
}


void loop()
{
}
```

A.8.4. Modulo Wifi ESP32

```
#include <WiFi.h>
```

```
#include <WebServer.h>

// Replace with your network credentials
const char* ssid = "youssid";
const char* password = "yourpassword";

WebServer server(80); // Web server on port 80

const int ledPin = 2; // Onboard LED pin (GPIO2 for ESP32)
bool ledState = false; // Track LED state

// HTML content for the web page
String HTMLPage() {
  String html = "<!DOCTYPE html><html>";
  html += "<head><title>ESP32 LED Control</title></head>";
  html += "<body style='text-align: center; font-family: Arial;'>";

  html += "<h1>ESP32 LED Control</h1>";
  html += ledState ? "<p>LED is <strong>ON</strong></p>" : "<p>LED
                       is <strong>OFF</strong></p>";
  html += "<a href='/on' style='padding: 10px 20px; background-color: green;
            color: white; text-decoration: none;'>Turn ON</a> ";
  html += "<a href='/off' style='padding: 10px 20px; background-color: red;
            color: white; text-decoration: none;'>Turn OFF</a>";

  html += "</body></html>";
  return html;
}

// Handle the root path "/"
void handleRoot() {
  server.send(200, "text/html", HTMLPage());
}

// Handle LED ON request
void handleLEDOn() {
  ledState = true;
  digitalWrite(ledPin, HIGH);
  server.send(200, "text/html", HTMLPage());
}

// Handle LED OFF request
void handleLEDOff() {
  ledState = false;
  digitalWrite(ledPin, LOW);
  server.send(200, "text/html", HTMLPage());
}
```

```
void setup() {
  Serial.begin(115200);

  // Initialize onboard LED pin as output
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW); // Start with LED off

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  Serial.print("Connecting to Wi-Fi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nConnected to Wi-Fi");
  Serial.println(WiFi.localIP());

  // Define server routes
  server.on("/", handleRoot);
  server.on("/on", handleLEDOn);
  server.on("/off", handleLEDOff);

  // Start the server
  server.begin();
  Serial.println("Server started");
}

void loop() {
  server.handleClient(); // Handle client requests
}
```

Apéndice B

Material complementario

B.1. Mapa completo del backend web

Este apéndice presenta el mapa completo de vistas y scripts del backend web del sistema. La figura complementa la descripción funcional realizada en el Capítulo 5.3,permitiendo visualizar la estructura de navegación y las dependencias internas entre archivos PHP.

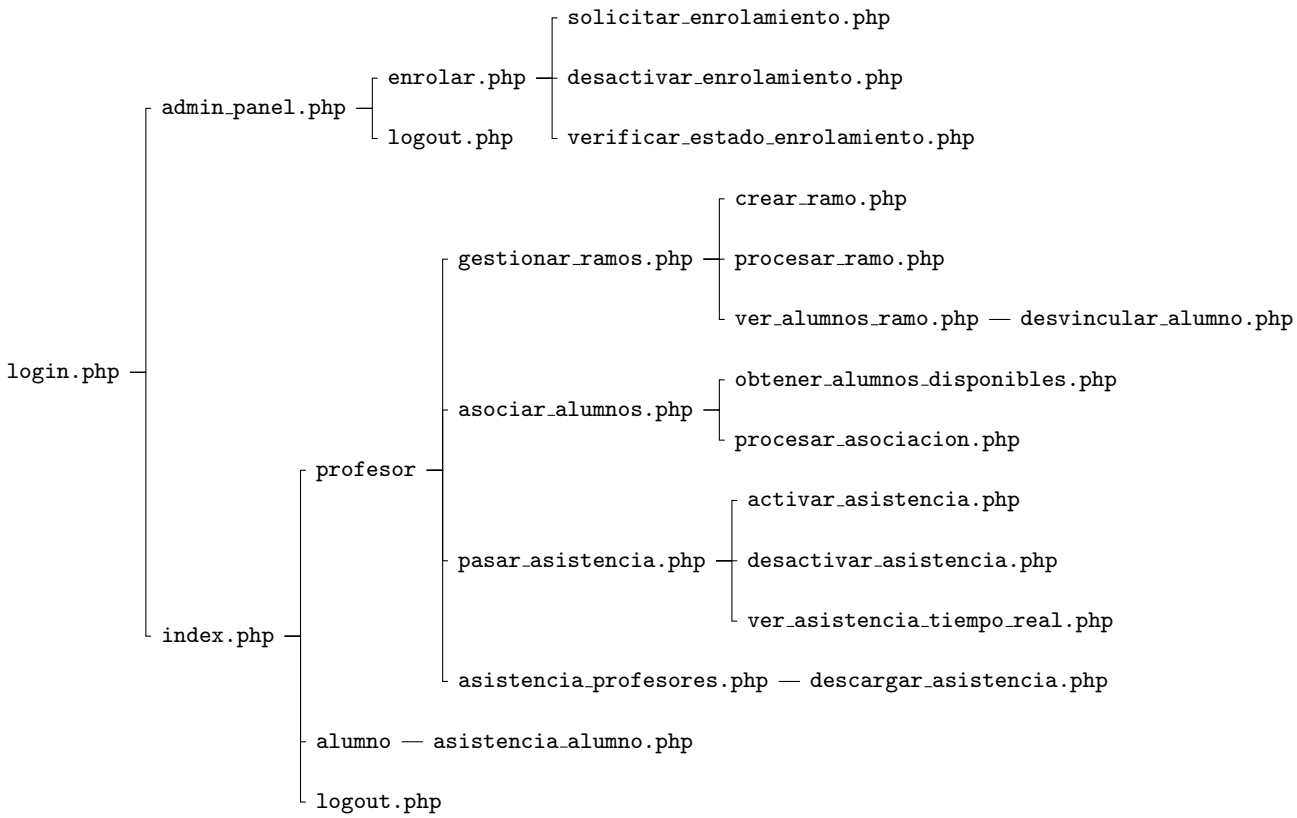


Figura B.1: Mapa completo de dependencias del backend web (vistas y scripts PHP).