UNIVERSIDAD DE MAGALLANES
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA EN COMPUTACIÓN

Diseño e Implementación de un Sistema de Control de Asistencia Automatizado con ESP32-CAM

Benjamín Vicente Miranda Benavides
2025

El presente Trabajo de Titulación ha sido aprobado con la siguiente calificación:

Nombre del estudiante: Benjamín Vicente Miranda Benavides
Trabajo de Título: _____
Examen de Título: _____
Nota Final: _____

Sr. Pedro Alberti Villalobos
Director Departamento
De Ingeniería En Computación

1 de Abril de 2025

UNIVERSIDAD DE MAGALLANES
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA EN COMPUTACIÓN

Diseño e Implementación de un Sistema de Control de Asistencia Automatizado con ESP32-CAM

Trabajo de titulación presentado en conformidad a los requisitos para obtener el título Ingeniería en Computación

Profesor Guía: Pedro Alberti Villalobos

Benjamín Vicente Miranda Benavides

2025

Resumen

El presente trabajo describe el diseño, desarrollo, implementación y validación de un sistema automatizado de registro de asistencia basado en el escaneo de códigos QR contenidos en la cédula de identidad chilena. La motivación principal surgió de la necesidad de superar las limitaciones de los métodos tradicionales de control de ingreso, como las planillas manuales o registros físicos, los cuales presentan problemas de eficiencia, precisión y seguridad.

La solución propuesta consiste en un dispositivo autónomo y compacto, construido sobre la plataforma ESP32-CAM, que captura imágenes de los carnets de identidad mediante una cámara integrada, ilumina el área de escaneo con un LED controlado por PWM, y transmite las imágenes a un sistema local para su procesamiento. La arquitectura del sistema se diseñó para operar en redes locales cerradas, garantizando la protección de los datos personales y evitando la dependencia de servicios en la nube.

El software de backend fue desarrollado utilizando el framework FastAPI, mientras que la interfaz de usuario fue implementada como una aplicación web accesible desde navegadores modernos. El sistema permite gestionar la información escaneada, visualizar historiales, consultar registros por fecha y restringir el acceso mediante autenticación por token. El procesamiento de los códigos QR se realiza mediante scripts desarrollados en Python, utilizando bibliotecas como Pyzbar y OpenCV para la decodificación y validación de datos.

Durante la fase de pruebas, el sistema demostró un rendimiento altamente satisfactorio: se logró un tiempo promedio de respuesta de 0.7 segundos por escaneo, sin fallas operativas en sesiones continuas de hasta cuatro horas. El diseño físico del dispositivo permitió una experiencia de uso intuitiva, con una curva de aprendizaje prácticamente nula, y obtuvo reacciones positivas por parte de los usuarios evaluadores.

El informe también detalla los antecedentes tecnológicos, la fundamentación teórica del uso de microcontroladores, visión artificial y códigos QR, así como la evolución del diseño desde un prototipo inicial voluminoso hasta una solución final optimizada y eficiente. Se destacan los aprendizajes obtenidos durante el proceso, incluyendo la importancia de considerar la ergonomía física en los dispositivos y el valor de un diseño centrado en el usuario.

En conclusión, este proyecto demuestra la viabilidad de implementar soluciones de control de asistencia eficientes, seguras y de bajo costo utilizando tecnologías de código abierto. Su diseño modular, adaptable y autónomo lo posiciona como una opción aplicable a diversos entornos institucionales, especialmente aquellos con recursos limitados o sin conectividad

constante a internet. Se proponen, además, recomendaciones para futuras mejoras, como la integración de funcionalidades inalámbricas adicionales y mejoras en la carcasa del dispositivo.

Índice

CAPÍTULO 1

INTRODUCCIÓN

Introducción

1.1. Introducción General

El **registro de asistencia** es una práctica fundamental en diversas instituciones académicas, centros de salud y organismos gubernamentales, ya que permite llevar un control adecuado sobre las personas que acceden a sus instalaciones. Este tipo de registro es esencial para mantener un seguimiento preciso de estudiantes, trabajadores, pacientes o visitantes, contribuyendo a una **gestión eficiente, segura y ordenada** de los flujos de ingreso y permanencia en un determinado recinto. Según diversas experiencias institucionales y normativas de control de acceso, este procedimiento facilita la trazabilidad de personas, la planificación de recursos humanos y la **respuesta oportuna en situaciones de emergencia** [11, 13].

No obstante, los métodos tradicionales de registro de asistencia presentan una serie de **limitaciones** que afectan su eficiencia y confiabilidad. En muchas instituciones, el control de ingreso se basa en listas físicas o planillas donde los usuarios deben anotar manualmente su nombre y su número de identificación, como el RUT. Este enfoque manual conlleva una serie de inconvenientes, tales como **errores de digitación** , pérdida de documentos, retrasos en la verificación de datos y vulnerabilidad ante **manipulaciones o falsificaciones**. Además, requiere que el personal encargado dedique una cantidad significativa de tiempo a la recopilación, procesamiento y verificación de la información, lo que puede derivar en una sobrecarga administrativa y afectar la productividad del establecimiento.

Con el avance de la tecnología y la creciente necesidad de automatizar procesos administrativos, han surgido diversas soluciones digitales para optimizar la gestión del registro de asistencia. Entre ellas se encuentran los sistemas basados en tarjetas RFID, reconocimiento facial o escaneo de códigos QR. Este último ha cobrado relevancia por su simplicidad, bajo costo e integración con plataformas móviles y de escritorio. En este contexto, se desarrolló un sistema automatizado basado en la lectura de códigos QR contenidos en los carnets de identidad vigentes de cada persona, los cuales son emitidos bajo un nuevo formato establecido por el Registro Civil chileno [5]. El escaneo se realiza mediante una cámara ESP32-CAM, que captura el código QR para extraer automáticamente los datos de identificación, almacenándolos en una base de datos local y visualizándolos mediante una interfaz web [10, 7, 4].

Aunque esta solución ha demostrado ser efectiva en la reducción de errores y en la optimización del tiempo de registro, su implementación inicial presentó ciertos desafíos relacionados con la portabilidad, el tamaño del equipo y su facilidad de integración en distintos espacios

laborales. En particular, el sistema requería un conjunto de dispositivos relativamente voluminosos y dependía de hardware que, si bien cumplía con su función, no era lo suficientemente compacto ni versátil para adaptarse a entornos donde el espacio es un recurso limitado. Pero en las fases finales del proyecto se va mejorando el prototipo mediante el desarrollo de un sistema más portátil, eficiente y adaptable, utilizando tecnologías de bajo costo y fácil integración, como la plataforma Arduino y el módulo ESP32-CAM [2, 22]. De esta manera, se busca optimizar el proceso de registro de asistencia sin comprometer la seguridad, confiabilidad y accesibilidad del sistema.

1.2. Antecedentes y Motivación

El control de asistencia y acceso a establecimientos es una necesidad creciente en diversos sectores, especialmente en ámbitos laborales y educativos. La implementación de métodos más eficientes y seguros no solo mejora la gestión administrativa, sino que también contribuye a la seguridad y al cumplimiento de normativas internas.

Los sistemas tradicionales han sido utilizados durante décadas, pero con la creciente digitalización de los procesos administrativos, han quedado obsoletos frente a soluciones más automatizadas. Algunas instituciones han optado por sistemas biométricos, tarjetas de proximidad o software en la nube para optimizar el control de asistencia. Sin embargo, estos métodos pueden ser costosos, requerir infraestructura compleja o depender de conectividad constante y muy estable a internet, lo que limita su aplicación en ciertos contextos.

El desarrollo del sistema de registro de asistencia basado en códigos QR surgió como una alternativa eficiente y accesible para reducir las limitaciones de los métodos manuales. No obstante, la implementación inicial debe evidenciar oportunidades de mejora en términos de portabilidad y facilidad de uso. El sistema no debía depender de equipos relativamente grandes, ya que restringiría su movilidad y dificultaría su despliegue en diferentes entornos.

Ante esta situación, la motivación principal de esta investigación es diseñar el sistema mediante la incorporación de tecnologías compactas y versátiles. La plataforma Arduino y el módulo ESP32-CAM ofrecen una solución de bajo costo, con capacidad para integrar sensores y conectividad inalámbrica, lo que permite desarrollar un dispositivo ligero, autónomo y adaptable a distintos escenarios [10, 2, 17].

1.3. Objetivos

El objetivo general es diseñar, desarrollar e implementar un piloto de sistema de registro automatizado de asistencia, portable y de fácil uso mediante la integración de tecnologías compactas y eficientes.

Para alcanzar este objetivo, se han definido los siguientes objetivos específicos:

1. **Incorporar un hardware pequeño y ligero:** Descartar cualquier hardware voluminoso por soluciones más compactas, como Arduino y ESP32-CAM, sin comprometer la funcionalidad del sistema.

2. **Garantizar la seguridad y confiabilidad de los datos:** Implementar protocolos de almacenamiento y respaldo de datos robustos para evitar pérdidas de información y mejorar la integridad del sistema [20].

3. **Diseñar la interfaz de usuario:** Desarrollar una interfaz intuitiva y adaptable a diferentes entornos para facilitar la interacción con el sistema [4, 11].

4. **Realizar pruebas exhaustivas:** Evaluar el desempeño del sistema en diferentes condiciones de uso, identificando áreas de mejora y aplicando ajustes según sea necesario.

1.4. Enfoque y Metodología

El desarrollo de este proyecto sigue un enfoque basado en la integración de tecnologías de bajo costo y fácil implementación, con el objetivo de crear una solución eficiente y accesible para el control de asistencia en distintos entornos.

Las etapas de desarrollo del sistema incluyen:

1. **Diseño del sistema:** Análisis de la arquitectura del sistema a desarrollar enfocándoze en la portabilidad y eficiencia.

2. **Desarrollo del hardware:** Integración de la plataforma Arduino y el módulo ESP32-CAM para obtener un resultado compacto y pequeño del dispositivo final y mejorar su autonomía [10].

3. **Implementación de la interfaz de usuario:** Creación de una interfaz intuitiva y personalizada que facilite la operación del sistema [4, 11].

4. **Pruebas y validación:** Evaluación del rendimiento del sistema en condiciones reales, garantizando su eficacia, precisión y durabilidad.

El enfoque metodológico combina el diseño de hardware y software con pruebas experimentales para asegurar que el sistema cumpla con los requisitos establecidos. Además, se realizarán iteraciones sobre el prototipo para optimizar su funcionalidad y mejorar la experiencia del usuario.

1.5. Alcances y Limitaciones

Este proyecto ha sido desarrollado con un enfoque inicial en el ámbito de la salud, particularmente en el registro de asistencia de pacientes. No obstante, su diseño flexible permite su adaptación a otros sectores, como instituciones académicas, empresas o entidades gubernamentales.

Dentro de sus alcances, el sistema ofrece una solución eficiente para el control de acceso, reduciendo errores y optimizando la gestión del tiempo. Sin embargo, presenta algunas limitaciones, entre ellas:

- **Dependencia de conexión a internet:** Aunque el sistema puede funcionar en una red local, su acceso a datos remotos requiere conectividad.
- **Requerimiento de hardware específico:** La implementación del sistema depende de dispositivos compatibles con la plataforma Arduino y ESP32-CAM.
- **Adaptabilidad a diferentes entornos:** Si bien el sistema es compacto, su implementación en espacios sin infraestructura tecnológica puede requerir ajustes adicionales.

A pesar de estas limitaciones, el proyecto establece una base sólida para futuras mejoras y expansiones, con la posibilidad de incorporar nuevas funcionalidades, como el reconocimiento facial [21] o la integración con bases de datos en la nube.

CAPÍTULO 2

MARCO TEÓRICO

Marco Teórico

2.1. Introducción

Este capítulo tiene como propósito presentar y describir cada uno de los elementos teóricos y tecnológicos que conforman el sistema de registro de asistencia desarrollado. Se abordan tanto los componentes de hardware como las herramientas de software utilizadas, explicando su función dentro del sistema y justificando su elección en base a criterios de eficiencia, portabilidad y bajo costo. Además, se contextualiza el uso de estas tecnologías en el desarrollo de soluciones automatizadas de asistencia, orientadas a instituciones de salud o entornos similares.

2.2. Componentes de Hardware

2.2.1. Módulo ESP32-CAM (Modelo AiThinker)

El **ESP32-CAM** es un microcontrolador basado en el chip ESP32, ampliamente utilizado para proyectos de visión artificial gracias a su cámara integrada OV2640, conectividad Wi-Fi y capacidad de procesamiento. En este proyecto se utilizó específicamente el modelo AiThinker, debido a su compatibilidad con múltiples entornos de desarrollo y la disponibilidad de documentación comunitaria.

El ESP32-CAM actúa como unidad de captura de imágenes del carnet de identidad, desde donde se extrae el código QR. Esta captura es enviada a través de una red Wi-Fi local al equipo que ejecuta el software de decodificación. Gracias a su tamaño reducido, bajo consumo energético y capacidades integradas, resultó ideal para este tipo de aplicación embebida.

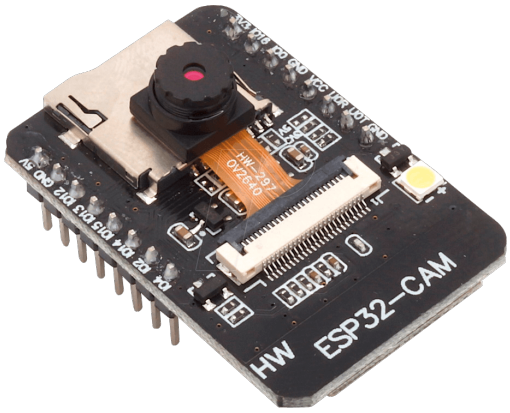

Figura 2.1: Módulo ESP32-CAM AiThinker utilizado en el proyecto.

2.2.2. Placa de Base ESP32-MB

Para facilitar la alimentación y programación del ESP32-CAM, se utilizó la placa **ESP32-MB**, la cual incluye un conector USB tipo C y un chip conversor serial USB-UART. Esta base permite prescindir de cables FTDI o pulsadores de reinicio, facilitando la carga del firmware y la conexión estable del sistema.

Figura 2.2: Módulo ESP32-MB utilizado en el proyecto.

2.2.3. Fuente de alimentación

El sistema fue alimentado mediante un **cargador de celular convencional** (5V - 3A), conectado por USB tipo C. Esta fuente es suficiente para alimentar el módulo incluso cuando el LED está encendido, asegurando estabilidad en las sesiones de captura prolongadas.

Figura 2.3: Cargador de celular utilizado

Figura 2.4: Cable USB-C para alimentación del ESP32-CAM

2.2.4. Carcasa personalizada

Una parte fundamental del diseño del sistema fue el desarrollo de una **carcasa personalizada** para alojar el módulo ESP32-CAM. Esta carcasa fue diseñada en el entorno **Tinkercad** y posteriormente impresa en 3D utilizando dos materiales distintos: primero **PLA** para prototipos de prueba, y finalmente **resina**, lo que permitió una mayor precisión y acabado estético.

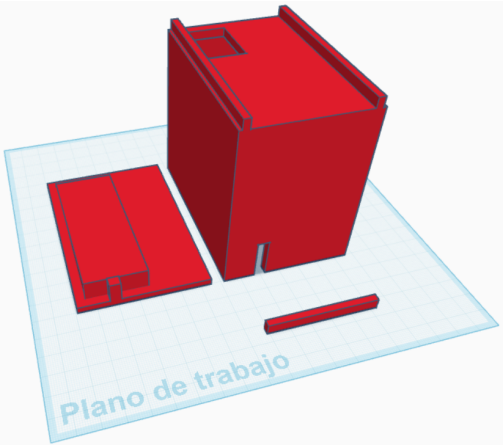

Figura 2.5: Carcasa personalizada diseñada en 3D.

2.3. Entornos y Lenguajes de Programación

2.3.1. Arduino IDE y C++

El firmware del módulo ESP32-CAM fue desarrollado en **C++** utilizando el entorno de desarrollo **Arduino IDE**. Este entorno permite programar microcontroladores de forma sencilla y flexible, con una amplia comunidad de soporte. Se utilizaron librerías como `WiFi.h` para la conexión a redes inalámbricas, y se configuraron parámetros como la calidad de imagen y la intensidad del LED mediante PWM. El código completo puede consultarse en el (véase Anexo A).

2.3.2. Visual Studio Code

Como entorno complementario, también se utilizó **Visual Studio Code (VS Code)** con extensiones de soporte para Python y Arduino, lo que facilitó el desarrollo simultáneo del backend y scripts de cliente en una sola plataforma.

2.4. Backend: FastAPI y Base de Datos

2.4.1. FastAPI

El servidor que administra la lógica del sistema fue desarrollado con **FastAPI**, un framework moderno y rápido para crear APIs en Python. FastAPI permite una implementación clara y segura de rutas protegidas mediante autenticación JWT, así como una rápida validación de datos gracias a **Pydantic**. Se utilizaron otras librerías como:

- **bcrypt:** Para el cifrado seguro de contraseñas.
- **JWT:** Para generar y validar tokens de acceso.
- **SQLAlchemy:** ORM para gestionar la base de datos relacional.

El código completo del backend puede consultarse en el (véase Anexo C).

2.4.2. Base de Datos: SQLite

Se utilizó **SQLite** como sistema de base de datos local, dada su facilidad de uso, ligereza y compatibilidad con entornos sin conexión a internet. Esta base contiene tablas para administrar usuarios, cámaras, horarios y registros de asistencia. La elección de SQLite está alineada con la filosofía del proyecto: mantener un sistema autónomo y funcional sin depender de servidores externos.

2.5. Frontend: HTML, JavaScript y Bootstrap

2.5.1. Estructura del sitio

La interfaz de usuario fue desarrollada como un sitio web local, accesible mediante navegador. Está compuesta por múltiples vistas: inicio de sesión, administración de cámaras, horarios y visualización del historial de pacientes. La interfaz fue desarrollada en **HTML y JavaScript**, apoyándose en el framework **Bootstrap 5** para lograr un diseño responsivo y moderno. Algunas capturas de la interfaz se presentan en el (véase Anexo F).

2.6. Scripts del Cliente

2.6.1. Librerías de procesamiento

Los scripts de procesamiento de imágenes y decodificación de códigos QR fueron desarrollados en **Python**, utilizando las siguientes librerías destacadas:

- **pyzbar:** Para decodificar códigos QR desde imágenes.
- **cv2 (OpenCV):** Para procesar imágenes capturadas.
- **numpy:** Para manipular matrices de píxeles.
- **re:** Para validación de patrones y limpieza de datos.

Estos scripts se encargan de capturar la imagen desde la ESP32-CAM, identificar el código QR del carnet y extraer datos como el nombre y RUT, los cuales son enviados al backend mediante solicitudes HTTP. El código se detalla en el (véase Anexo E).

2.7. Comunicación entre Componentes

2.7.1. Protocolo HTTP en red local

La comunicación entre la ESP32-CAM y el computador que ejecuta los scripts de cliente se realiza mediante **HTTP** sobre una **red Wi-Fi local**. La ESP32-CAM actúa como servidor web que entrega imágenes al cliente, mientras que el cliente procesa la información y la envía al backend a través de endpoints protegidos. Esta arquitectura garantiza que el sistema funcione de forma autónoma sin depender de conexión a internet externa, cumpliendo con los estándares de seguridad al operar en redes internas controladas (véase Figura 2.6).

Figura 2.6: Arquitectura general del sistema basado en red local

CAPÍTULO 3

DESARROLLO

Desarrollo del Tema

3.1. Planteamiento del problema

El registro de asistencia es una actividad esencial en diversas instituciones y organizaciones, ya que permite llevar un control ordenado y documentado de las personas que acceden a un determinado recinto. Este tipo de control es fundamental no solo desde una perspectiva administrativa, sino también desde una óptica de seguridad, trazabilidad y cumplimiento normativo. En entornos como centros de salud, instituciones educativas y oficinas gubernamentales, saber con exactitud quién ha ingresado, a qué hora, y por cuánto tiempo ha permanecido dentro de las instalaciones puede ser crítico para la toma de decisiones, la gestión de recursos humanos, la planificación de actividades y, especialmente en contextos de emergencia, para la localización oportuna de personas.

Tradicionalmente, el registro de asistencia se ha efectuado mediante métodos manuales, tales como listas en papel o planillas físicas, donde los usuarios deben ingresar sus datos de forma escrita. Aunque estos métodos han sido ampliamente utilizados y en muchos casos aún persisten, presentan una serie de desventajas significativas: son susceptibles a errores humanos, tales como digitación incorrecta o ilegibilidad; implican un proceso lento y poco práctico; requieren de espacio físico para almacenamiento de documentos; y pueden ser objeto de manipulación o falsificación de información, lo cual compromete la integridad del registro. Además, demandan tiempo y esfuerzo del personal encargado, generando una carga administrativa que puede impactar negativamente en la eficiencia global de la institución.

En respuesta a estas limitaciones, han emergido soluciones tecnológicas orientadas a digitalizar el proceso de control de asistencia. Sin embargo, muchas de estas soluciones, como los sistemas biométricos, tarjetas inteligentes o plataformas basadas en la nube, implican costos elevados de implementación y mantenimiento, así como la necesidad de infraestructura tecnológica avanzada o conectividad constante a internet, condiciones que no siempre están disponibles en todos los entornos, especialmente en regiones con recursos limitados o en organizaciones pequeñas [13].

El proyecto en cuestión surge a partir de la necesidad de diseñar un sistema de registro de asistencia que supere las limitaciones mencionadas, ofreciendo una solución automatizada, confiable y de bajo costo, que sea adaptable a distintos contextos. La implementación inicial consistió en un sistema de escaneo de códigos QR contenidos en la cédula de identidad nacional chilena, aprovechando que este documento incluye un código QR con datos relevantes

para la identificación de cada persona [5]. Este sistema logró capturar automáticamente los datos de los usuarios y almacenarlos en una base de datos local, facilitando su posterior consulta a través de una interfaz gráfica desarrollada con tecnologías web [4].

No obstante, si bien el presente sistema a implementar cumplía con su propósito funcional, su diseño fisico presentaba obstáculos relevantes en cuanto a portabilidad y escalabilidad. El conjunto de equipos requeridos para el funcionamiento del sistema no debían ser voluminosos para su uso en espacios reducidos o situaciones donde la movilidad era un factor clave. Esto evidenció la necesidad de una reingeniería del sistema, orientada a reducir el tamaño del hardware, mejorar su autonomía y garantizar una integración más sencilla y versátil. Para ello se requería el uso de tecnologías portátiles, pequeñas, duraderas y de bajo costo (Anexos A y B).

3.2. Desarrollo teórico

Para abordar el diseño del sistema de registro de asistencia, es necesario comprender los fundamentos teóricos en los que se sustenta la propuesta tecnológica. El proyecto se basa en el uso de microcontroladores y dispositivos de visión artificial, integrados a través de plataformas de desarrollo embebido como Arduino y ESP32-CAM [2, 22, 10]. Estas tecnologías permiten implementar soluciones robustas, eficientes y de bajo costo para aplicaciones que requieren captura de datos, conectividad inalámbrica y procesamiento básico en tiempo real.

3.2.1. Tecnologías embebidas y microcontroladores

Un microcontrolador es un circuito integrado que contiene un procesador, memoria y periféricos de entrada/salida programables, permitiendo la ejecución de tareas específicas sin requerir un sistema operativo complejo. La plataforma Arduino, ampliamente utilizada en entornos educativos y de prototipado rápido, proporciona una interfaz accesible y versátil para programar y controlar dispositivos electrónicos [17]. En particular, placas como Arduino UNO, Nano o Mega permiten desarrollar soluciones que integren sensores, actuadores y comunicación serial o inalámbrica.

Por su parte, el módulo **ESP32-CAM** representa una evolución en términos de capacidad y funcionalidad. Basado en el chip ESP32, este dispositivo cuenta con conectividad Wi-Fi y Bluetooth integrada, y añade una cámara OV2640 que permite capturar imágenes en tiempo real [10]. Estas características hacen del ESP32-CAM una solución ideal para aplicaciones de visión artificial, monitoreo remoto, y automatización basada en reconocimiento de patrones o códigos, como los QR. La implementación de este módulo en el presente proyecto puede revisarse en detalle en el Anexo A.

3.2.2. Códigos QR como mecanismo de identificación

El código QR (Quick Response) es un tipo de código de barras bidimensional que puede almacenar una cantidad considerable de información en una pequeña área. Su estructura de matriz de módulos blancos y negros permite una lectura rápida y precisa desde diversos ángulos y dispositivos. En el contexto chileno, la cédula de identidad emitida por el Registro Civil incluye un código QR que contiene información codificada, como el RUT, nombres, fecha

de nacimiento y número de documento [5]. Esta característica se aprovecha para automatizar el proceso de identificación de personas sin necesidad de ingreso manual de datos.

3.2.3. Procesamiento de imágenes y decodificación de QR

Para que el sistema pueda leer el código QR desde la cédula de identidad, es necesario implementar un flujo de procesamiento de imagen que incluya la captura, análisis y decodificación del patrón QR. El módulo ESP32-CAM, junto con librerías especializadas como OpenCV o pyzbar, permite realizar este proceso localmente o mediante el envío de imágenes a un servidor que ejecute la decodificación [21, 9]. Esta etapa requiere condiciones mínimas de iluminación, enfoque y posicionamiento del documento, lo cual plantea desafíos técnicos en cuanto al diseño físico del dispositivo que contendrá la cámara, y a la calibración del sistema para garantizar una lectura rápida y precisa.

3.2.4. Almacenamiento y gestión de datos

Una vez obtenida la información del código QR, esta debe ser almacenada en una base de datos que permita su consulta, análisis y exportación. Dado que el sistema está diseñado para operar en entornos que pueden carecer de conectividad constante, se optó por un modelo de almacenamiento local utilizando bases de datos ligeras como SQLite [20]. Esto garantiza independencia del sistema frente a la conectividad y permite una recuperación eficiente de la información. El manejo de esta base de datos está documentado en el código del backend (véase Anexo B).

3.2.5. Interfaz de usuario y visualización de datos

La interfaz de usuario (UI) cumple un papel fundamental en la experiencia de uso del sistema. Una UI bien diseñada debe permitir al operador visualizar en tiempo real los registros de ingreso, realizar búsquedas por nombre o RUT, generar reportes y acceder a funciones administrativas básicas. En el marco del presente proyecto, se optó por desarrollar una interfaz accesible mediante navegador web, compatible con distintos dispositivos, lo que facilita su despliegue y acceso desde diferentes puntos dentro del establecimiento [4, 11]. El diseño de esta interfaz puede revisarse en detalle en el Anexo C.

3.2.6. Seguridad y protección de datos personales

La gestión de datos personales impone obligaciones en términos de confidencialidad, integridad y disponibilidad de la información. El sistema propuesto incorpora mecanismos de validación y autenticación para limitar el acceso a la base de datos, y utiliza métodos de encriptación para el almacenamiento seguro de los datos [7]. Asimismo, se busca cumplir con los principios establecidos por la Ley N.º 19.628 sobre Protección de la Vida Privada, vigente en Chile, garantizando que los datos recolectados sean utilizados exclusivamente para los fines declarados y con las debidas salvaguardas legales.

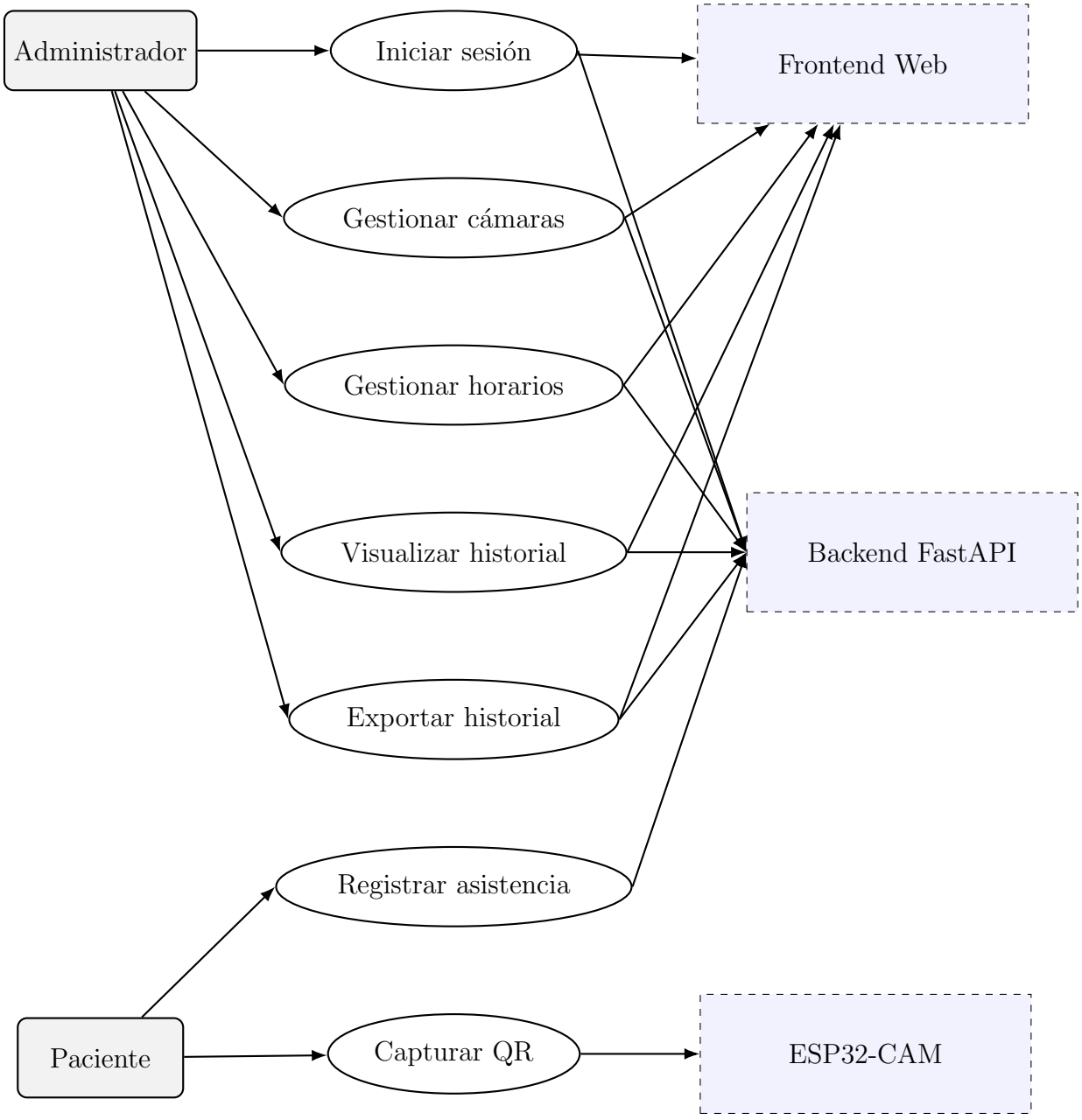

Figura 3.1: Diagrama de Casos de Uso del Sistema Completo

3.3. Desarrollo experimental

El desarrollo experimental de este proyecto implicó una serie de etapas clave orientadas a materializar un sistema funcional, confiable y práctico para el registro automatizado de asistencia mediante la lectura de códigos QR integrados en la cédula de identidad chilena [5]. Esta fase del proyecto se caracterizó por la experimentación con distintas configuraciones de hardware, diseño de prototipos físicos, implementación de software de control, y pruebas funcionales en entornos simulados, todo con el objetivo de alcanzar una solución optimizada y adecuada para su despliegue en contextos reales como centros de salud.

3.3.1. Diseño inicial del prototipo

El desarrollo del prototipo de un sistema automatizado para el registro de asistencia mediante escaneo de códigos QR implicó, en su fase inicial, un enfoque exploratorio que combinó diferentes componentes de hardware con el fin de encontrar una solución equilibrada entre funcionalidad, costo y facilidad de implementación. En este contexto, la primera versión del prototipo se concibió utilizando una arquitectura basada en la combinación de un microcontrolador **Arduino Nano** y un módulo de cámara **ESP32-CAM**.

El razonamiento inicial detrás del uso del Arduino Nano se centraba en distribuir responsabilidades funcionales entre los dos dispositivos: por un lado, se buscaba que el Arduino cumpliera el rol de **controlar el encendido de la cámara**, gestionar ciertos eventos de entrada/salida y servir como **puente de comunicación entre la cámara y la computadora**. Esta idea se sustentaba en experiencias previas con proyectos basados en Arduino, donde este microcontrolador había demostrado ser útil para tareas de control y monitoreo de bajo nivel. Por otro lado, se pretendía que el módulo ESP32-CAM se limitara a la captura de imágenes, en una especie de arquitectura cooperativa entre ambos dispositivos.

Para esta configuración inicial se utilizó una **fuente de alimentación directa desde un enchufe**, conectando los cables de alimentación y control a los pines correspondientes tanto del Arduino como de la cámara. Se empleó una **protoboard** para facilitar la conexión entre los componentes, evitando soldaduras para mantener la posibilidad de realizar ajustes rápidos y flexibles durante la etapa de pruebas.

No obstante, esta primera configuración se enfrentó rápidamente a **una serie de dificultades técnicas importantes**. En primer lugar, el uso de una protoboard —si bien práctico

en etapas iniciales— implicó una conexión física poco robusta. El cableado, al no estar soldado ni fijado de forma permanente, se volvió propenso a falsos contactos, desconexiones accidentales y pérdidas intermitentes de señal o alimentación. Esta situación fue especialmente problemática considerando que el sistema debía operar de forma estable y continua para garantizar un registro de asistencia confiable.

En segundo lugar, la **alimentación eléctrica compartida entre el Arduino Nano y la ESP32-CAM resultó inestable**. El consumo de energía del módulo de cámara es relativamente alto durante la inicialización y la captura de imágenes, lo que provocaba caídas de voltaje en el circuito, afectando negativamente al rendimiento del Arduino. Como resultado, el microcontrolador no lograba procesar correctamente las instrucciones, presentando errores durante la comunicación con la cámara y afectando la respuesta general del sistema.

Adicionalmente, el rol que se había asignado al Arduino —como controlador y puente— demostró ser innecesariamente redundante. A pesar de las buenas intenciones iniciales, pronto se evidenció que el módulo ESP32-CAM por sí solo era capaz de cumplir con todas las funciones esperadas, incluyendo el procesamiento de imágenes, la conexión a la red y la entrega de datos al sistema receptor. Este descubrimiento fue clave para reorientar el desarrollo del prototipo.

Como resultado de estos problemas y del análisis funcional del sistema, se decidió **descartar completamente el uso del Arduino Nano** en la arquitectura final. Esta decisión marcó un punto de inflexión importante en el proyecto. Si bien puede considerarse un error inicial, también representó una **valiosa lección sobre el potencial del ESP32-CAM como plataforma autónoma**. La experiencia sirvió para reconocer las verdaderas capacidades del módulo, lo que permitió no solo simplificar la arquitectura del sistema, sino también mejorar su estabilidad y reducir su tamaño considerablemente [10, 22].

La nueva arquitectura se reconfiguró alrededor de la **integración directa del módulo ESP32-CAM con un módulo base ESP32-MB**, una placa de desarrollo diseñada para facilitar el acceso a los pines del ESP32 y proveer una conexión sólida mediante un único **cable USB tipo C**. Este cambio fue decisivo en términos de robustez y practicidad. La conexión física ya no dependía de cables jumper ni de conexiones temporales en una protoboard, sino que se establecía de forma firme, segura y estable, como si se tratara de un dispositivo enchufable común. Esta simplicidad en la integración redujo drásticamente la posibilidad de fallas por mal contacto, y permitió una alimentación constante y confiable directamente desde un puerto USB de computador o desde una fuente externa de 5V y 3A [2].

Desde el punto de vista funcional, esta reestructuración no solo mejoró la estabilidad del sistema, sino que permitió **optimizar el código** cargado en la cámara. Al eliminar la intermediación del Arduino, el código de control pudo ser reducido y enfocado exclusivamente en la lógica de captura y transmisión de imágenes desde el ESP32-CAM, lo que **mejoró la velocidad de respuesta y redujo el margen de error en la comunicación**. La nueva versión, cargada mediante el entorno Arduino IDE, incluía la configuración de la red Wi-Fi, la inicialización de la cámara, el control del LED integrado (para garantizar una iluminación constante) y la puesta en marcha de un servidor web en el puerto 80 (véase Anexo A) para entregar las imágenes al sistema central.

Uno de los aspectos más valorados del nuevo diseño fue la mejora en términos de **compacidad, limpieza y eficiencia estructural**. La eliminación del Arduino Nano liberó espacio dentro del prototipo y permitió avanzar rápidamente al diseño de una carcasa más pequeña, más estética y más fácil de instalar en entornos reales. Mientras que el primer diseño debía albergar múltiples placas y un entramado de cables relativamente complejo, la nueva arquitectura redujo el conjunto a una **única placa compacta**, unida a su base ESP32-MB, con un único cable USB tipo C como conexión principal. Este rediseño también facilitó enormemente la creación de la carcasa final mediante impresión 3D.

En retrospectiva, si bien el uso inicial del Arduino Nano puede considerarse una decisión errónea desde el punto de vista técnico, también fue una **etapa fundamental del proceso de aprendizaje y evolución del prototipo**. Experimentar con esa configuración permitió entender los límites de ciertos enfoques y, al mismo tiempo, valorar la autonomía del ESP32-CAM como solución embebida completa. Esta fase temprana del desarrollo fue clave para sentar las bases de una solución más refinada, minimalista y sólida, que hoy constituye el corazón del sistema automatizado de registro de asistencia.

3.3.2. Configuración del hardware y programación de la cámara

La correcta configuración del hardware y la programación del módulo **ESP32-CAM** representó una etapa clave en el desarrollo de este sistema de registro automatizado de asistencia. Dado que este componente actúa como el núcleo del dispositivo —encargado de capturar imágenes de las cédulas de identidad y transmitirlas para su procesamiento— fue esencial configurar adecuadamente tanto el entorno físico de funcionamiento como el software embebido que controla su operación. A continuación, se describe detalladamente el proceso

seguido y las decisiones técnicas adoptadas, las cuales permitieron consolidar un sistema funcional, eficiente y de fácil integración (véase Anexo A).

Conexión a red Wi-Fi y configuración flexible del SSID

El módulo ESP32-CAM fue programado para conectarse automáticamente a una red Wi-Fi de 2.4 GHz o 5 GHz, utilizando credenciales almacenadas en su memoria [10]. Inicialmente, el SSID y la contraseña de la red estaban codificados directamente en el firmware, lo que obligaba a modificar el código fuente en caso de querer cambiar de red. No obstante, para mejorar la experiencia del usuario final y facilitar la instalación en distintos entornos, se desarrolló una interfaz gráfica amigable que permite modificar estos parámetros sin necesidad de reprogramación (véase Anexo C).

Comunicación vía servidor HTTP embebido

El ESP32-CAM fue configurado para levantar un servidor HTTP en el puerto 80, el cual responde a solicitudes entrantes desde dispositivos conectados a la misma red local [10]. Este servidor implementa los endpoints mínimos necesarios para capturar imágenes en tiempo real. Aunque se evaluó la implementación de WebSockets, se optó por HTTP por su simplicidad y estabilidad, priorizando la funcionalidad durante la fase de prototipado.

Resolución de captura y calidad de imagen

Se seleccionó una resolución de 640x480 píxeles (VGA), ya que ofrecía un equilibrio entre claridad del código QR y velocidad de procesamiento. La compresión JPEG se fijó en un 80 % de calidad para mantener legibilidad sin afectar significativamente la transferencia. Esta configuración se ajustó tras pruebas empíricas que demostraron que calidades más bajas generaban errores de lectura del código QR [21].

Control del LED integrado

El LED blanco de alta intensidad incorporado en la ESP32-CAM fue controlado mediante modulación por ancho de pulso (PWM) en el pin GPIO4, regulando el brillo para evitar sobreexposición. La intensidad óptima se encontró en un valor de PWM igual a 7 (véase Anexo A). Este componente permitió prescindir de fuentes de luz externas, simplificando el diseño general del dispositivo.

Estabilidad térmica y operativa

Durante sesiones de uso continuo superiores a cuatro horas, el módulo se mantuvo estable, sin desconexiones ni sobrecalentamiento. La temperatura máxima medida fue de aproximadamente 55°C, valor completamente dentro de los rangos seguros para este tipo de hardware [10].

Alimentación y consumo energético

El sistema fue alimentado mediante un cable USB tipo C conectado a una fuente estándar de 5V y 3A. Aunque se consideró el uso de powerbanks para mejorar la portabilidad, se descartó debido a limitaciones de espacio y la necesidad de una fuente continua. La corriente consumida por el dispositivo fue estimada entre 160 mA en reposo y 310 mA durante la captura activa, lo que equivale a un consumo de potencia aproximado entre 0,8 W y 1,55 W, considerando una tensión de alimentación constante de 5V.

Programación y ajuste del firmware

El código fue desarrollado en C++ y cargado mediante el entorno Arduino IDE [2, 17]. Se utilizó la configuración base para la placa AiThinker, realizando ajustes mínimos como la inicialización del servidor HTTP, brillo del LED y calidad de la imagen (véase Anexo A). El uso de funciones estándar de la librería ESP32 asegura la portabilidad del sistema a otros módulos similares.

Usabilidad y retroalimentación para el usuario

Para mejorar la experiencia de uso, el sistema incluye indicadores visuales (LED) y señales acústicas desde el computador que ejecuta el software cliente, permitiendo validar el éxito de cada escaneo sin necesidad de pantallas o monitores (véase Anexo D). Esta solución ha demostrado ser intuitiva y efectiva incluso para usuarios sin conocimientos técnicos.

Pruebas bajo condiciones diversas de iluminación

Se realizaron pruebas en condiciones de iluminación baja, luz cálida y luz blanca, todas en interiores. El LED del dispositivo proporcionó iluminación suficiente para garantizar una

captura estable, y el diseño de la carcasa minimizó reflejos o sombras que pudieran interferir con la lectura del código QR (véase Anexo B).

Seguridad y justificación del entorno local

El sistema opera únicamente en redes locales, utilizando autenticación mediante tokens con expiración y sin exposición a Internet, lo que minimiza riesgos de interceptación [7]. Si bien no se implementó HTTPS, la restricción de acceso físico y la segmentación de red propias de las instituciones en que se implementará justifican esta decisión.

3.3.3. Desarrollo del software de captura y procesamiento

El desarrollo del software de captura y procesamiento constituye una de las piezas fundamentales del sistema automatizado de registro de asistencia, ya que es el componente encargado de orquestar la interacción entre el dispositivo ESP32-CAM, el análisis de los datos contenidos en los códigos QR y la posterior gestión de la información a través de un backend seguro y funcional.

Arquitectura general del sistema de captura

El sistema de captura ha sido implementado completamente en lenguaje **Python**, aprovechando su robustez, facilidad de integración con bibliotecas externas y versatilidad para la programación de tareas automáticas. El software se estructura en forma de módulos interconectados, cada uno con responsabilidades específicas:

- **Módulo de captura de imagen:** realiza solicitudes al ESP32-CAM a través de HTTP para obtener frames en tiempo real.
- **Módulo de decodificación de QR:** procesa cada imagen capturada para detectar y extraer información del código QR.
- **Módulo de interpretación de datos:** convierte el contenido del QR en una estructura legible, extrayendo específicamente el RUT.
- **Módulo de comunicación con backend:** utiliza los datos extraídos para consultar la base de datos mediante peticiones a la API desarrollada en FastAPI.

- **Módulo principal (main.py):** actúa como orquestador, ejecutando el sistema según el horario programado y gestionando los eventos de forma secuencial.

El sistema corre en segundo plano mediante un script ejecutable, sin requerir interacción directa del usuario. Al iniciar, establece conexión con la cámara, verifica la red, y comienza a operar en un ciclo de escaneo continuo mediante un bucle `while True`. Si se detecta un código QR válido, se detiene la captura para procesar los datos y reinicia el ciclo al terminar la operación.

Procesamiento y decodificación de códigos QR

Para la detección y decodificación de códigos QR se hace uso de las bibliotecas **OpenCV**, **NumPy**, **pyzbar** y **re** (expresiones regulares). El proceso inicia con la conversión de la imagen capturada a escala de grises y una mejora de contraste, optimizando la visibilidad del patrón QR. Una vez procesada, se escanea la imagen con `pyzbar.decode()` y se extrae el contenido textual del código.

El sistema identifica el RUT a partir de dicho contenido utilizando expresiones regulares, lo convierte al formato oficial con puntos y guion, y posteriormente ejecuta una consulta al backend para obtener el nombre completo del paciente. Esta modularidad permite mantener cada parte del sistema desacoplada y fácil de mantener.

Cabe destacar que, gracias al diseño físico de la caja y al correcto posicionamiento del carnet, la lectura ha demostrado ser extremadamente precisa, sin necesidad de recortar la imagen ni definir una región específica. El sistema identifica de forma automática la ubicación del código QR dentro del campo visual.

Backend local con FastAPI

El backend fue desarrollado con **FastAPI** [7], una tecnología moderna que permite construir APIs **RESTful** [12] de alto rendimiento, con una sintaxis clara, validaciones automáticas y excelente integración con **SQLAlchemy** [19] para manejo de bases de datos. El backend corre en el mismo equipo que el software cliente, garantizando así un entorno completamente **local** que no depende de conectividad externa ni servicios en la nube.

Entre sus funcionalidades principales se encuentran:

- Registro y autenticación de administradores mediante **JWT** [6].

- Gestión de cámaras, incluyendo nombre, credenciales de red y horarios de funcionamiento.
- Registro y consulta de pacientes asociados a cada cámara.
- Asignación de horarios específicos por día y hora a cada dispositivo.
- Control de accesos a rutas seguras usando tokens **Bearer** [15].
- Validación de contraseñas seguras mediante **bcrypt** [3].

Los tokens generados tienen una duración de 60 minutos y son almacenados localmente para permitir el uso continuo del sistema sin necesidad de relogueo constante. El backend está organizado por modelos y esquemas **Pydantic** [8] que permiten validar la entrada de datos y evitar errores comunes en el ingreso de información.

Base de datos y estructura de información

El sistema utiliza una base de datos local **SQLite** [20], ideal para proyectos medianos por su bajo consumo de recursos, velocidad de acceso y facilidad de distribución sin necesidad de un servidor dedicado. La base de datos incluye varias tablas: `administradores`, `camaras`, `horarios` y `pacientes`, todas relacionadas entre sí para facilitar consultas y filtrados por horarios o dispositivos.

La información registrada se limita a datos esenciales: nombre y RUT, vinculados a la cámara que escaneó y al horario correspondiente. Por diseño, **las imágenes capturadas no se almacenan**, lo que reduce el uso de espacio en disco y refuerza la privacidad de los usuarios.

Experiencia de uso

El sistema fue diseñado para ser completamente invisible al usuario final. El cliente simplemente presenta su cédula frente al lector. Un sonido indica que la cámara está activa y conectada a red, y otro sonido suave indica que el código fue leído correctamente. No se requiere interacción con mouse ni teclado.

El frontend del sistema, que se abordará en otra sección, permite consultar el historial de escaneos y exportar los registros en formato Excel.

Análisis de diseño

El software de captura y procesamiento cumple un rol fundamental dentro del sistema de asistencia automatizada, garantizando una integración fluida entre hardware, procesamiento de datos y gestión de información. Su diseño modular, funcionamiento autónomo y operación completamente local lo convierten en una solución confiable, segura y fácil de escalar para su aplicación en diferentes entornos institucionales.

3.3.4. Diseño y fabricación del prototipo físico

El diseño y fabricación del prototipo físico fue una etapa crítica y muy compleja del proyecto, en la cual se buscó no solo proteger los componentes electrónicos, sino también optimizar la experiencia del usuario final. Desde sus primeras versiones hasta el modelo definitivo, el dispositivo pasó por múltiples iteraciones que respondieron a criterios de funcionalidad, estética, facilidad de instalación y robustez.

Diseño conceptual y evolución del prototipo

Desde el comienzo, el dispositivo fue concebido con una **forma rectangular**, sin bordes redondeados, tanto por motivos estructurales como funcionales. Este diseño fue elegido por su facilidad de montaje en muros verticales, y porque guarda una coherencia visual con la forma del carnet de identidad chileno, lo que facilita que los usuarios comprendan intuitivamente cómo posicionar su documento para el escaneo.

El diseño fue realizado íntegramente en **Tinkercad** [1], y tras cinco iteraciones sucesivas, se alcanzó un modelo definitivo. Cada nueva versión permitió reducir el tamaño del dispositivo, mejorar el aprovechamiento del espacio interno, incorporar guías físicas para el posicionamiento del carnet y optimizar la orientación de la cámara para lograr una captura más estable e iluminada.

Durante las versiones iniciales, se posicionó la cámara de diferentes formas (en un lateral, en la parte inferior), pero tras pruebas y observaciones, se optó por una **orientación vertical ascendente**, con la cámara fija en la base y mirando hacia arriba. Este enfoque facilita que los usuarios simplemente apoyen su cédula sobre la parte superior del dispositivo, sin necesidad de inclinarse ni manipularlo.

Primeras versiones en PLA

Las primeras cajas fueron impresas utilizando **filamento PLA** en las impresoras 3D disponibles en la universidad. Aunque útiles para probar medidas y posiciones, estas versiones presentaron diversas limitaciones. Entre los principales inconvenientes se encontraron:

- **Deformaciones** por temperaturas elevadas durante el uso.
- **Falta de precisión** en los detalles finos.
- **Poca resistencia estructural**, lo que generaba inestabilidad al fijar los componentes.

El tiempo de impresión en PLA oscilaba entre **9 y 11 horas para el cuerpo principal** y entre **5 y 6 horas para la tapa**, con un peso estimado de aproximadamente un kilogramo. Estas versiones permitieron aprender a diseñar piezas más compactas, reforzar puntos clave y agregar relieves en la estructura interna que ayudaran a centrar la cédula de identidad frente a la cámara.

Fabricación final en resina

El diseño definitivo fue impreso en **resina blanca con propiedades similares a la madera**, mediante un servicio externo de impresión 3D. Aunque se desconoce la marca exacta de resina utilizada, se identificaron sus características principales: **textura semirrugosa, resistencia a impactos, mayor rigidez y acabado visual superior** al PLA.

La resina ofreció además mayor estabilidad dimensional, permitiendo que las piezas calzaran con más precisión y sin necesidad de ajustes mecánicos complejos. La estructura fue diseñada en dos partes:

- **Cuerpo principal**, que contiene la cámara, los rieles y el área de posicionamiento del carnet.
- **Base desmontable**, que se fija con tornillos y permite acceder al interior para tareas de mantenimiento.

El proceso de fabricación en resina tomó **aproximadamente dos días**: uno para la impresión total de las piezas y otro para el secado completo, lijado y pintado. La cámara fue adherida a la base mediante un adhesivo no conductor de alta fijación, evitando desplazamientos o tambaleos durante el uso.

Funcionalidad, ergonomía y montaje en muro

El dispositivo fue diseñado para ser **montado directamente en muro**, utilizando tornillos. Se recomienda una **altura entre 1,5 y 1,6 metros**, pensada para ser cómoda para la mayoría de los usuarios, especialmente pacientes en clínicas u hospitales. Para asegurar un correcto posicionamiento del documento, el diseño incluye un **riel horizontal y guías verticales** adaptadas tanto para el carnet antiguo como el nuevo, facilitando una lectura precisa del código QR.

El cable de alimentación es el único visible y **sale por un orificio trasero**, quedando sujeto mediante un **clip interno para cables USB tipo C**, que previene torceduras y prolonga la vida útil del cable. La entrada USB queda **oculta en el interior del dispositivo**, garantizando seguridad y limpieza visual.

Consideraciones de durabilidad, mantenimiento y seguridad

El prototipo final fue construido con criterios de durabilidad. La caja ofrece **protección frente a golpes** y mantiene la cámara en una posición segura y estática. Aunque la cámara puede reemplazarse fácilmente en caso de fallo, el principal desafío no es su valor ni disponibilidad (es un componente ampliamente distribuido), sino los tiempos de envío, que pueden extenderse entre **1 y 2 semanas**.

El diseño facilita tareas de mantenimiento: la base puede desmontarse sin herramientas especializadas y se puede acceder al módulo ESP32-CAM sin riesgo de dañar el sistema. No se incorporó ventilación activa debido a que las pruebas demostraron que **la temperatura operativa de la cámara es baja**, y no requiere enfriamiento adicional.

En conjunto, el diseño físico del prototipo responde a criterios de **ergonomía, funcionalidad, estética y facilidad de mantenimiento**, siendo una solución robusta, profesional y adecuada para su uso en entornos reales como centros de salud.

3.3.5. Evaluación funcional y pruebas experimentales

La validación funcional del sistema se llevó a cabo mediante un conjunto de pruebas experimentales diseñadas para simular las condiciones reales de operación en un entorno clínico. Estas pruebas fueron ejecutadas tanto en el hogar del desarrollador como en las instalaciones de la universidad, permitiendo analizar el rendimiento del sistema desde múltiples

ángulos: estabilidad del hardware, precisión del escaneo, respuesta del software, ergonomía del dispositivo y experiencia del usuario final.

Entorno y metodología de prueba

Las pruebas se realizaron en dos contextos distintos: un entorno doméstico controlado y espacios académicos que permitieron emular el flujo y la dinámica de un centro de salud. En ambos casos, el dispositivo fue instalado sobre mesas, escritorios y finalmente en muros, replicando la configuración que se espera en una instalación definitiva. Se contó con la participación de **siete personas de distintas edades**, quienes presentaron su cédula de identidad chilena ante el lector, permitiendo evaluar la respuesta del sistema en función de la variabilidad de los documentos y usuarios.

El flujo de personas no fue estructurado, imitando una dinámica espontánea y realista. La duración promedio de cada sesión de prueba fue de aproximadamente una hora, y se repitieron varias veces a lo largo del desarrollo, superando en total los **200 escaneos únicos**, lo cual permitió reducir progresivamente el margen de error mediante ajustes iterativos al diseño físico y al código fuente.

Evaluación de compatibilidad con distintos documentos

Todas las pruebas se centraron exclusivamente en la **cédula de identidad chilena vigente**, tanto en su formato anterior como en el nuevo modelo emitido por el Registro Civil a partir del 16 de diciembre de 2024. A pesar de que ambos carnets mantienen la calidad del código QR, presentan una diferencia clave: la ubicación del código. Mientras que el carnet anterior posiciona el QR en la esquina superior derecha, el nuevo diseño lo mantiene a la derecha pero lo desplaza hacia el centro vertical del documento.

Este cambio obligó a rediseñar el sistema de guías físicas en la carcasa: se eliminó el riel vertical único y se reemplazó por **dos guías marcadas**, una para cada tipo de carnet. El sistema no requiere distinguir entre formatos de carnet, ya que el algoritmo de lectura es lo suficientemente robusto para interpretar correctamente el QR sin importar su altura, siempre que este se encuentre dentro del campo de visión de la cámara.

Desempeño del sistema y velocidad de lectura

El desempeño del sistema fue evaluado principalmente mediante logs del software, los cuales registran el momento de captura exitosa del QR. En fases iniciales, el tiempo promedio de lectura oscilaba entre **2 y 3 segundos**. Sin embargo, gracias a mejoras en la iluminación, enfoque, estabilidad del soporte y guías de posicionamiento, el tiempo fue reducido a aproximadamente **0.5 segundos en las pruebas finales**.

Las mejores condiciones para la lectura fueron aquellas con uso de **luz LED blanca a baja potencia**, estabilidad física proporcionada por el riel horizontal y una posición ergonómica que permite apoyar el carnet sin necesidad de sostenerlo en el aire. No se observaron diferencias de desempeño entre el carnet nuevo y el anterior, más allá del requerimiento de ajustar la posición.

No se reportaron errores de lectura de RUT ni fallas en la detección del QR, salvo algunas dificultades en el proceso de obtención del nombre completo cuando se trataba de pacientes jóvenes cuyos datos no estaban bien registrados. Estos errores se consideraron menores y se tratarán en futuras versiones del software.

Iluminación y condiciones ambientales

Durante las primeras pruebas se trabajó únicamente con **luz natural**, pero esta demostró ser inconsistente, afectando la nitidez de las capturas. Posteriormente se implementó iluminación artificial mediante **LED blanco**, la cual resultó mucho más confiable. El sistema también fue probado con todas las luces apagadas, funcionando perfectamente solo con el LED integrado del ESP32-CAM, el cual resultó ser **crítico para el éxito del escaneo**.

Se realizaron pruebas con diferentes intensidades de iluminación del LED. Una potencia demasiado alta provocaba reflejos que impedían la lectura del QR, mientras que una potencia muy baja afectaba la visibilidad. Finalmente, se estableció un nivel de brillo óptimo que permite una lectura rápida y sin errores.

Experiencia del usuario final

La retroalimentación de los participantes fue mayoritariamente positiva. La mayoría encontró el sistema **intuitivo y rápido**, y bastaron unas pocas instrucciones verbales para que los usuarios comprendieran el procedimiento. La posición del carnet sobre la caja resultó cómoda para adultos y personas de baja estatura, aunque aún no se han realizado pruebas

con personas en silla de ruedas. Las ventajas del diseño modular del dispositivo permitirán ajustar su altura en instalaciones futuras.

Los usuarios valoraron especialmente la **rapidez del sistema**, tanto en la lectura como en el procesamiento. Las señales sonoras integradas facilitaron aún más la experiencia, informando de manera discreta y efectiva cuándo la cámara estaba lista y el código había sido escaneado correctamente.

Iteraciones posteriores y ajustes

Tras cada sesión de pruebas, se realizaron mejoras incrementales. Entre ellas se encuentran:

- Ajustes en la carcasa para soportar mejor ambos tipos de carnet.
- Inclusión de guías verticales duales.
- Mejoras en los sonidos de alerta.
- Modificaciones menores al código para gestionar errores y mejorar la experiencia de usuario.

En términos de rendimiento técnico, la cámara operó durante **más de 4 horas continuas sin fallas** ni signos de sobrecalentamiento. Se tomaron lecturas térmicas con un termómetro infrarrojo portátil, registrando una temperatura de funcionamiento estable.

Durante las fases tempranas sí se detectaron errores en el software (por ejemplo, en la lectura del backend o conexión inicial), pero todos fueron solucionados con pruebas posteriores. Hoy el sistema opera con alta fiabilidad y sin errores registrados durante las últimas sesiones de prueba.

Aunque aún no se ha realizado una prueba formal con documentos no válidos (por ejemplo, sin código QR), se planea abordar esta situación en pruebas futuras para garantizar un manejo adecuado de casos excepcionales.

En conclusión, las pruebas funcionales y experimentales han demostrado que el sistema es **estable, rápido, intuitivo y adaptable**, cumpliendo con todos los requisitos para ser implementado en entornos reales como centros de salud. Todas las pruebas realizadas han sido consideradas críticas para validar el sistema, y los resultados obtenidos justifican plenamente su potencial de implementación a mayor escala.

3.3.6. Estado actual y perspectivas

Actualmente, el sistema automatizado de registro de asistencia se encuentra en una etapa avanzada de desarrollo, habiendo superado exitosamente todas las fases de validación en entornos simulados. El siguiente paso consiste en evaluar su funcionamiento en un entorno real, con condiciones de uso cotidiano, flujos reales de usuarios y requerimientos operativos concretos. Para ello, se ha iniciado un proceso de colaboración con el **Hospital Regional de Punta Arenas**, específicamente con el área de **Kinesiología**.

El contacto ha sido establecido a través de una trabajadora del área, **Susan**, quien ha demostrado un interés notable en el proyecto. Durante las reuniones sostenidas hasta ahora, se ha valorado positivamente la naturaleza **intuitiva, digital y fácil de operar** del sistema, tanto desde la perspectiva de los usuarios (pacientes) como del personal administrativo. No se ha requerido hasta el momento ningún informe formal, pero sí se han hecho presentaciones del funcionamiento del prototipo y se han registrado observaciones para futuras mejoras.

Instalación en el hospital y condiciones de implementación

La instalación piloto se realizará en la **recepción del área de Kinesiología**, junto a la puerta interna que da acceso al pabellón. De esta forma, los pacientes podrán registrar su asistencia **justo antes de ingresar** al tratamiento, presentando su cédula de identidad sobre el lector. El sistema utilizará el **computador de recepción** como terminal para ejecutar el software y gestionar la base de datos local, lo que elimina la necesidad de llevar equipos externos.

Uno de los requisitos clave pendientes es la disponibilidad de una **red Wi-Fi privada**, que el hospital se encuentra tramitando actualmente. Esta red es indispensable para permitir la comunicación entre el ESP32-CAM y el software local. Una vez que Susan confirme la disponibilidad de dicha red, se procederá con la instalación definitiva del sistema. En caso de que la aprobación se retrase, se considera la posibilidad de rediseñar la carcasa para adaptarla a posibles nuevas condiciones.

Se cuenta ya con **documentación técnica básica** para facilitar el uso del sistema por parte del personal del hospital. Esta incluye instrucciones para iniciar el software, verificar la conexión del dispositivo y operar el backend desde el navegador. Además, se preparan instrucciones visuales y sonoras que hacen que el sistema sea completamente accesible incluso para usuarios sin conocimientos técnicos.

Perspectivas de mejora y evolución futura

Una vez instalado, se espera obtener retroalimentación directa del personal, en especial sobre la **interfaz de usuario** y la **usabilidad general del sistema**. Estas observaciones se recopilarán mediante reuniones breves o mensajería directa con el personal, con el fin de aplicar mejoras progresivas. Entre las funcionalidades futuras sugeridas se incluye la opción de que el sistema genere **informes automáticos** a partir de los registros capturados, por ejemplo, para análisis de demanda mensual, control estadístico o reportes contables.

La escalabilidad del sistema está contemplada desde su diseño: cualquier otra unidad del hospital que cuente con una recepción y un computador podría implementar el mismo sistema sin modificaciones relevantes. Las únicas limitantes técnicas actuales son la **disponibilidad de conexión Wi-Fi privada** y un **equipo de escritorio básico** que ejecute el software. No se recomienda su uso en áreas de urgencia debido a la naturaleza del flujo rápido y no programado de pacientes.

Escalabilidad y adaptabilidad a otros sectores

Más allá del ámbito hospitalario, el sistema ha despertado interés en otras instituciones, como el **SLEP (Servicio Local de Educación Pública)**, quienes han considerado su aplicación en el **registro de asistencia estudiantil**. Si bien esta posibilidad presenta desafíos adicionales en términos de infraestructura tecnológica (como la conectividad y disponibilidad de computadoras en las escuelas), la adaptabilidad del sistema lo convierte en una solución viable para múltiples sectores.

Asimismo, su implementación podría beneficiar a **empresas, centros de formación, eventos masivos, municipalidades y otras entidades** que requieran registrar de manera ágil y segura la asistencia de personas. Para estos casos, sería necesario únicamente ajustar algunos aspectos del **frontend** o del **backend** (principalmente en el modelado de usuarios y presentación de datos), pero el hardware, flujo de lectura y sistema de autenticación podrían mantenerse prácticamente sin cambios.

Consideraciones para la implementación institucional

La instalación en un entorno real como el hospital también implica considerar **aspectos administrativos y técnicos adicionales**. Aunque el sistema está diseñado para operar **en una red local cerrada y protegida con autenticación JWT**, lo que minimiza

significativamente los riesgos de exposición de datos, es probable que el área de informática del hospital solicite una revisión técnica del sistema o documentación de seguridad para autorizar la instalación definitiva.

Los desafíos principales que se anticipan son:

- Disponer de una red Wi-Fi estable y segura.
- Asignar un computador operativo con permisos de administrador.
- Contar con una aprobación oficial del área correspondiente (Susan o su jefatura).

A pesar de estos requerimientos, el sistema presenta claras **ventajas operativas**: su bajo costo, facilidad de uso, independencia de servicios externos y rapidez de respuesta lo convierten en una herramienta altamente funcional y replicable.

En conclusión, el proyecto se encuentra en una etapa clave: la transición desde la validación simulada hacia la validación en un entorno hospitalario real. Esta instancia permitirá no solo confirmar su viabilidad técnica y operativa, sino también **abrir la puerta a su expansión futura** en distintos sectores, consolidando su valor como solución tecnológica accesible, segura y adaptable.

3.4. Descripción detallada del código fuente

Con el fin de proporcionar una visión integral del funcionamiento interno del sistema, esta sección está dedicada a explicar en profundidad la lógica, estructura y funciones principales de los distintos componentes de software desarrollados para este proyecto. El código fuente se encuentra organizado en cuatro bloques fundamentales, cada uno con un rol específico dentro del sistema completo:

1. **Código en Arduino IDE para la configuración del módulo ESP32-CAM**

   Este código es responsable de establecer la conectividad Wi-Fi del dispositivo, inicializar la cámara, ajustar la resolución y calidad de imagen, y habilitar la captura mediante solicitudes HTTP. También incluye el control del LED integrado para asegurar una iluminación adecuada durante el escaneo.

2. **Código en Python para la comunicación con la cámara ESP32-CAM**

   Este script ejecutable se encarga de solicitar imágenes periódicamente al módulo ESP32-CAM, procesarlas en tiempo real y detectar códigos QR. Al detectar un QR válido, extrae el RUT, lo formatea, y lo envía al backend local para su validación y almacenamiento.

3. **Código del Backend (FastAPI)**

   Desarrollado con Python y el framework FastAPI, este backend gestiona la base de datos local, administra los usuarios, cámaras y horarios, y proporciona autenticación mediante JWT. Expone una serie de endpoints protegidos para registrar y consultar pacientes, así como para manejar configuraciones del sistema.

4. **Código del Frontend**

   Interfaz visual accesible desde un navegador local, conectada al backend. Permite al personal ver el historial de registros, consultar pacientes por fecha y exportar los datos en formato Excel. El diseño busca simplicidad y eficiencia para usuarios sin conocimientos técnicos.

A continuación, se procederá a detallar individualmente cada uno de estos bloques de código, comenzando con el desarrollado en Arduino IDE para el módulo ESP32-CAM.

3.4.1. Código en Arduino IDE para la ESP32-CAM

El código desarrollado en Arduino IDE para el módulo ESP32-CAM cumple una función esencial dentro del sistema: inicializa la cámara, la conecta a la red Wi-Fi local y establece un servidor HTTP que permite recibir peticiones para la captura de imágenes en formato JPG. Además, incorpora el control de un LED integrado utilizando modulación por ancho de pulso (PWM), lo cual garantiza una iluminación adecuada durante la captura, especialmente en condiciones de baja luz.

A continuación se describe detalladamente la funcionalidad del código:

- **Inclusión de librerías:**

  Se importan las bibliotecas necesarias, `WifiCam.hpp` para la gestión de la cámara y `WiFi.h` para establecer la conectividad inalámbrica.

- **Definición de credenciales Wi-Fi:**

  Se definen dos constantes `WIFI_SSID` y `WIFI_PASS` con el nombre de la red y su contraseña respectivamente. Estas credenciales permiten conectar automáticamente el módulo ESP32-CAM a la red local.

- **Configuración del servidor:**

  Se inicializa un servidor HTTP en el puerto 80, mediante el objeto `WebServer`, que será responsable de recibir solicitudes entrantes y responder con capturas de imagen.

- **Configuración del LED:**

  Se define el pin GPIO4 como pin de control del LED integrado. También se especifica el canal PWM a utilizar, la frecuencia (5000 Hz), la resolución (8 bits) y la intensidad de brillo (valor de 7 sobre 255). Esta configuración permite controlar la potencia lumínica con precisión, evitando sobreexposición.

- **Función `setup()`:**

  Esta función se ejecuta una única vez al inicio del programa y realiza las siguientes tareas:

  - Inicializa el puerto serial a 115200 baudios para fines de depuración y monitoreo.

- Intenta establecer la conexión Wi-Fi utilizando las credenciales definidas. Si la conexión falla, el dispositivo espera 5 segundos y luego se reinicia automáticamente para intentar de nuevo.
- Configura y habilita el canal PWM asociado al LED, inicialmente dejándolo apagado (`ledcWrite(PWM_CHANNEL, 0)`).
- En un bloque aislado (para facilitar lectura y mantenimiento), se configura la cámara:
  - Se establece como modelo el AiThinker, que es compatible con el ESP32-CAM.
  - Se selecciona una resolución de 1024x768 píxeles, que permite un buen equilibrio entre calidad de imagen y tamaño del archivo.
  - Se ajusta la compresión JPEG a un valor de 80, lo que proporciona buena calidad visual con un tamaño aceptable.
  - Si la cámara no se inicializa correctamente, se imprime un mensaje de error y se reinicia el sistema.
- Una vez iniciada correctamente la cámara, se enciende el LED con el nivel de brillo especificado (bajo), para asegurar iluminación constante durante las capturas.
- Finalmente, se imprime por consola la dirección IP local asignada al dispositivo, y se llama a `addRequestHandlers()` (función encargada de asociar rutas del servidor a acciones específicas) para luego iniciar el servidor.

- **Función `loop()`:**

  Esta función se ejecuta de manera continua y se encarga de gestionar cada petición recibida por el servidor HTTP mediante `server.handleClient()`.

Este código garantiza que la cámara esté en estado de escucha activa, lista para responder a peticiones desde el software cliente mediante HTTP. El uso de PWM para el control del LED permite ajustar la intensidad de forma precisa, reduciendo la posibilidad de reflejos y mejorando la calidad de la imagen capturada.

La arquitectura del código favorece la estabilidad del sistema, incorporando reinicios automáticos en caso de fallos de conexión o inicialización, lo cual mejora la confiabilidad en entornos reales. Esta implementación proporciona un equilibrio óptimo entre simplicidad, eficiencia y funcionalidad para el propósito de escanear cédulas de identidad chilenas mediante código QR.

3.4.2. Código Python para la comunicación y procesamiento de imágenes

El segundo componente crítico del sistema está compuesto por un conjunto de scripts desarrollados en Python, cuya función es establecer comunicación con el módulo ESP32-CAM, procesar las imágenes capturadas en tiempo real, detectar códigos QR, extraer el RUT y posteriormente consultar un servicio web para obtener el nombre completo del paciente. Todo este flujo se encuentra modularizado en archivos independientes, lo que mejora la mantenibilidad del código y permite realizar modificaciones parciales sin comprometer la estructura general del sistema. A continuación se describen los módulos involucrados, partiendo del flujo principal.

1. main.py — Coordinador del sistema de escaneo

Este script constituye el módulo principal que orquesta la ejecución del sistema. Su lógica se basa en la consulta de los horarios activos para el día actual y la ejecución condicional del ciclo de escaneo de pacientes solo si la hora actual se encuentra dentro del horario autorizado. El flujo general de `main.py` es:

- Importa funciones desde los módulos `rut.py`, `nombre.py` y `consultas_backend.py`.
- Utiliza el identificador de cámara para consultar los horarios del día mediante `obtener_horario()`.
- Si existe un horario válido, se ejecuta un ciclo `while` que:
  - Captura un RUT escaneado desde la cámara.
  - Obtiene el nombre asociado al RUT desde un servicio web.
  - Registra los datos mediante una llamada al backend con `registrar_paciente()`.
  - Espera 5 segundos antes de permitir un nuevo escaneo para evitar saturación del sistema.

Esta estructura modular favorece la ejecución autónoma del script y permite escalar o ajustar su comportamiento con facilidad.

2. rut.py — Captura y decodificación de códigos QR

Este módulo encapsula toda la lógica relacionada con la adquisición y procesamiento de imágenes desde el ESP32-CAM. Sus principales funcionalidades incluyen:

- Establecer conexión con la cámara a través de una URL de streaming MJPEG.
- Leer los frames y aplicar correcciones de perspectiva si es necesario.
- Mejorar el contraste y brillo con `convertScaleAbs()` para optimizar la lectura del código QR.
- Detectar códigos QR mediante la biblioteca `pyzbar`.
- Extraer el RUT utilizando expresiones regulares que buscan el patrón `RUN=xxxxx&` dentro del contenido del QR.
- Formatear el RUT extraído al formato chileno estándar con puntos y guion.

La función `ejecutar_camara()` permanece en ejecución hasta detectar un código QR válido. Una vez encontrado, detiene la captura, retorna el RUT y libera los recursos de la cámara.

3. nombre.py — Consulta del nombre desde el RUT

Este script se encarga de realizar una solicitud a un sitio web público (`nombrerutyfirma.com`) utilizando un scraper que evade sistemas de protección como Cloudflare (gracias a la librería `cloudscraper`).

- Dado un RUT, el módulo construye la URL de búsqueda correspondiente.
- Realiza la solicitud HTTP al servidor remoto.
- Utiliza XPath y la librería `lxml` para extraer el nombre del paciente desde el HTML de respuesta.
- Si el nombre es encontrado, lo retorna en formato texto. De lo contrario, retorna un mensaje de error.

Este paso permite complementar los datos obtenidos desde el QR con un identificador legible por humanos para ser almacenado y visualizado en el sistema.

4. consultasbackend.py - Integración con la API local (FastAPI)

Este módulo agrupa todas las funciones relacionadas con la interacción entre el sistema local y el backend implementado en FastAPI. Sus funciones clave son:

- `obtener_token()` – Solicita un token JWT al backend para autenticar todas las peticiones.
- `get_headers()` – Devuelve los encabezados necesarios para las solicitudes autenticadas.
- `registrar_paciente()` – Envia los datos de nombre, RUT y hora de ingreso al endpoint `/paciente/`. Si la operación es exitosa, se despliega una ventana emergente utilizando Tkinter para notificar al usuario.
- `obtener_horario()` – Consulta el endpoint correspondiente para recuperar los horarios programados para la cámara en uso, filtrando por día.
- `obtener_camara()` – Permite acceder a la configuración de una cámara específica, en caso de ser necesario para validaciones adicionales.

Este módulo permite encapsular toda la lógica de autenticación y consulta de datos, manteniendo el resto del sistema desacoplado de la lógica de red.

En conjunto, estos cuatro módulos permiten al sistema operar de forma autónoma, ejecutando ciclos de escaneo inteligentes, extrayendo datos relevantes, validando la identidad del paciente y registrando los eventos con mínima intervención del usuario. La estructura modular del código favorece tanto su mantenimiento como su futura ampliación con nuevas funcionalidades.

3.4.3. Código Backend desarrollado en FastAPI

El backend del sistema fue desarrollado utilizando el framework FastAPI, una herramienta moderna y eficiente para el desarrollo de APIs RESTful en Python. Esta capa del sistema es responsable de manejar la lógica de negocio, la autenticación, el control de acceso, la gestión de usuarios, horarios, cámaras y registros de pacientes.

Estructura General El archivo principal del backend integra varios módulos y bibliotecas relevantes:

- `FastAPI` para la creación de endpoints y manejo de dependencias [7].
- `SQLAlchemy` como ORM para interactuar con una base de datos SQLite.
- `Pydantic` para validar los modelos de entrada y salida de datos.
- `bcrypt` y `jwt` para autenticación y manejo seguro de credenciales.
- `CORS` para permitir la comunicación con un frontend alojado en otro dominio o puerto.

Base de Datos y ORM El sistema utiliza una base de datos local SQLite (`test.db`) y define cuatro **entidades principales** dentro de su esquema de datos:

- `Administrador`: Usuarios con permisos para acceder a rutas protegidas.
- `Camara`: Representa cada dispositivo físico, incluyendo su nombre, red WiFi y URL de streaming.
- `Horario`: Asociado a una cámara, define los tramos horarios permitidos para el escaneo.
- `Paciente`: Representa a una persona que ha sido registrada con nombre, RUT, hora y día de ingreso.

Estos modelos están enlazados por relaciones `ForeignKey` y permiten consultas complejas mediante `SQLAlchemy`.

Seguridad y Autenticación El sistema cuenta con un sistema de autenticación basado en `OAuth2PasswordBearer` y tokens **JWT (JSON Web Tokens)**:

- Se utiliza una **clave criptográfica privada** definida por el desarrollador, junto con el algoritmo `HS256` para la generación y validación de tokens.
- Los usuarios se autentican mediante un endpoint `/token` que devuelve un token de acceso válido por 60 minutos.
- Todas las rutas críticas están protegidas mediante el decorador `Depends(get_current_user)`, que verifica que el token JWT sea válido y no haya expirado.
- Las contraseñas de los administradores son encriptadas usando `bcrypt`.

Endpoints Principales

- **Administradores**: Permiten crear, consultar y eliminar usuarios administradores.
- **Cámaras**: Permiten registrar un nuevo dispositivo, consultarlo, eliminarlo o actualizar su configuración (incluyendo nombre, red WiFi y URL).
- **Horarios**: Permiten asignar bloques horarios a las cámaras para controlar cuándo están habilitadas para registrar pacientes.
- **Pacientes**: Esta sección maneja las operaciones más relevantes del sistema. Permite:
  - Registrar un paciente si no ha sido ingresado ese mismo día.
  - Consultar todos los pacientes registrados.
  - Eliminar pacientes por su RUT.
  - Consultar los últimos seis pacientes registrados por cámara.
  - Consultar todos los pacientes registrados en una fecha específica.

Todas estas funcionalidades utilizan la validación y manipulación de datos a través de esquemas definidos con `Pydantic`, lo que garantiza consistencia y seguridad.

Características Especiales

- Se integró la biblioteca `tkinter` en la comunicación cliente (desde Python) para mostrar notificaciones emergentes cada vez que un paciente es registrado correctamente.
- Todos los registros de pacientes incluyen hora y día de ingreso, lo que permite generar reportes diarios o por bloques horarios desde el frontend.
- La seguridad del sistema se ve reforzada al ejecutarse localmente en una red cerrada, con autenticación integrada y sin exposición pública en la web.

Consideraciones para Producción Aunque el backend ha sido diseñado para un entorno local controlado, su arquitectura permite escalar a entornos más complejos:

- Es posible cambiar la base de datos a PostgreSQL o MySQL sin grandes cambios.
- Se puede adaptar para ejecutarse en servidores en la nube usando HTTPS.
- Las políticas de autenticación pueden ampliarse con control de roles o renovación automática de tokens.

En resumen, el backend actúa como núcleo del sistema, validando entradas, asegurando integridad de datos y respondiendo de forma eficiente a las solicitudes del frontend y del sistema de escaneo Python. Su diseño modular y seguro lo convierte en una base sólida para cualquier mejora o expansión futura del proyecto.

3.4.4. Código del Frontend

El sistema cuenta con una interfaz web desarrollada completamente en HTML, CSS, JavaScript y Bootstrap, orientada a la gestión y visualización de datos de asistencia. A continuación se describen las cuatro vistas que componen el frontend, destacando su funcionalidad, estructura y relación con el backend.

1. Inicio de Sesión (`index.html`): Esta es la vista de entrada del sistema, donde los usuarios deben autenticarse con un nombre de usuario y una contraseña.

- **Formulario**: Incluye campos de texto para el nombre y contraseña, validados con HTML5 y gestionados por JavaScript.

- **Autenticación**: Al enviar el formulario, se realiza una petición POST a la ruta `/token` del backend usando `fetch()` con `application/x-www-form-urlencoded`.
- **Seguridad**: Si las credenciales son válidas, se guarda el `access_token` en `localStorage` y se redirige al usuario a la vista `camaras.html`. En caso contrario, se muestra un mensaje de error.

2. Gestión de Cámaras (`index.html`): Permite visualizar, editar y consultar las cámaras del sistema.

- **Tabla de cámaras**: Muestra información como ID, sector, nombre de red Wi-Fi y contraseña.
- **Modal de edición**: Se utiliza para actualizar los datos de una cámara mediante una petición `PUT` al backend.
- **Consulta de pacientes**: Al hacer clic en "Ver", se abre un modal con los últimos seis pacientes registrados por la cámara seleccionada, debido a la consulta hacia el endpoint del backend.
- **Eliminación**: Es posible eliminar registros individuales de pacientes con un botón y confirmación previa.
- **Protección**: La vista verifica si el token JWT está presente; si no, redirige a `index.html`.

3. Gestión de Horarios (`index.html`): Permite crear, editar y eliminar los horarios en que cada cámara estará activa para registrar pacientes.

- **Tabla de horarios**: Muestra ID, día, hora de inicio, término e ID de cámara.
- **Modales**:
  - **Crear horario**: Abre un formulario para ingresar nuevos registros mediante petición `POST`.
  - **Editar horario**: Llenado dinámico mediante JavaScript, con petición `PUT` para guardar cambios.

- **Eliminación**: Se confirma la acción antes de realizar la petición `DELETE` al backend.
- **Horarios automáticos**: Los selectores de hora se generan automáticamente de 30 en 30 minutos.

4. Historial de Pacientes (`index.html`): Vista orientada a la consulta y exportación del historial de pacientes ingresados por fecha.

- **Selector de fecha**: Utiliza `bootstrap-datepicker` para permitir seleccionar una fecha y consultar los datos correspondientes.
- **Tabla dinámica**: Se llena mediante petición GET a `/pacientes/por-dia/{fecha}`.
- **Ordenamiento**: Permite ordenar por nombre u hora mediante clics en los encabezados.
- **Exportación**: Incluye un botón que convierte la tabla HTML a Excel usando la biblioteca `xlsx`.
- **Autenticación**: Requiere token JWT válido almacenado en `localStorage` para acceder a los datos.

Características comunes del Frontend:

- **Autenticación JWT**: Todas las vistas, salvo `index.html`, están protegidas mediante token almacenado en el navegador.
- **Diseño responsive**: Gracias a Bootstrap, el sistema es compatible con diferentes tamaños de pantalla.
- **Modularidad**: Cada archivo HTML cumple una función específica y hace uso de fetch para comunicarse con el backend.
- **Interacción fluida**: El uso de modales para editar y crear entidades mejora la experiencia de usuario sin necesidad de recargar la página.
- **Estilo uniforme**: Gracias al uso de Bootstrap y Bootstrap Icons, todas las vistas comparten una estética coherente y moderna.

Gracias a esta estructura y funcionalidad, el frontend complementa perfectamente al backend, facilitando el uso del sistema para administradores y personal de recepción sin necesidad de conocimientos técnicos avanzados.

3.5. Exposición y Discución de los resultados

La etapa de exposición de resultados constituye uno de los hitos más importantes en el proceso de validación de un prototipo tecnológico. En este caso particular, los resultados obtenidos del sistema automatizado de registro de asistencia con escaneo de cédulas de identidad han sido evaluados en entornos simulados y controlados, permitiendo verificar el cumplimiento de los objetivos planteados y la efectividad del sistema en condiciones próximas a su uso real.

3.5.1. Rendimiento del sistema

El rendimiento del sistema fue evaluado a través de múltiples sesiones de prueba realizadas tanto en el domicilio del desarrollador como en instalaciones de la universidad. Durante la fase final de prototipado, se realizaron aproximadamente 70 escaneos con los últimos diseños de hardware y software. El tiempo promedio de lectura del código QR fue de tan solo 0,7 segundos por escaneo, un valor altamente competitivo en comparación con sistemas similares de registro. Cabe destacar que con los diseños finales se logró una tasa de éxito del 100 % en la lectura de los códigos QR, sin registrar fallos ni omisiones.

No se evidenciaron diferencias significativas entre la lectura del QR presente en la antigua cédula de identidad chilena y el nuevo formato emitido desde diciembre de 2024. Esto se debe, en gran medida, a los ajustes realizados en la carcasa del prototipo, que incorpora guías visuales para ambos tipos de cédula. En cuanto a la iluminación, se comprobó que la luz natural no era suficiente, especialmente debido al diseño cerrado de la carcasa. No obstante, la luz LED integrada en la ESP32-CAM demostró ser completamente eficiente incluso en entornos con escasa o nula iluminación externa.

3.5.2. Experiencia de usuario

El dispositivo fue probado por siete usuarios de distintas edades, quienes manifestaron una alta satisfacción con la experiencia de uso. Las principales fortalezas destacadas fueron

la rapidez del sistema y su facilidad de uso. Gracias al diseño físico del dispositivo, que dirige de manera intuitiva el posicionamiento de la cédula sobre el escáner, los usuarios tardaron menos de 30 segundos en entender cómo operar el sistema sin necesidad de instrucciones previas.

Antes del diseño final, se recibió retroalimentación útil por parte de los usuarios, particularmente en lo que respecta a la incorporación de guías más amplias para facilitar el posicionamiento del carnet, lo que fue implementado exitosamente. Aunque no se ha probado aún con personas con discapacidades motoras significativas, se considera que la simplicidad del sistema permitirá una adaptación adecuada.

3.5.3. Estabilidad, robustez y fiabilidad

Las pruebas de resistencia del sistema incluyeron sesiones prolongadas de entre 3 y 4 horas continuas, sin que se observaran caídas del sistema ni errores críticos. Durante estas sesiones, se mantuvo la estabilidad tanto del hardware como de la conexión entre el módulo ESP32-CAM y el software Python. La temperatura máxima registrada fue de 50 ^{o}C, muy por debajo del límite de operación de la ESP32-CAM (125 ^{o}C), y el encapsulado del chip contribuyó eficazmente a la disipación del calor.

Para garantizar la fiabilidad del sistema, se implementaron múltiples mecanismos de control de errores en el software. Estos incluyen mensajes para usuarios no autenticados, restricciones para evitar el registro duplicado de un paciente en el mismo día, alertas en caso de uso fuera del horario definido y validaciones de funcionamiento de la cámara.

3.5.4. Iteraciones y mejoras significativas

El proyecto atravesó una evolución considerable desde su primera versión hasta el estado actual. En términos de hardware, las versiones iniciales incorporaban componentes como un Arduino Nano, una batería externa (powerbank) y una lámpara LED adicional, además de una carcasa de gran tamaño. Esta configuración fue depurada significativamente, quedando reducida a un diseño mucho más compacto basado únicamente en la ESP32-CAM alimentada por cable USB-C. Esta simplificación no solo mejoró la eficiencia energética y estética del sistema, sino que facilitó su portabilidad y montaje en muros.

En cuanto al software, se abandonó un sistema rudimentario que solo procesaba códigos QR y generaba registros en hojas de cálculo, para dar paso a una arquitectura robusta

compuesta por un Backend y un Frontend personalizados. Esta nueva estructura opera de manera totalmente local, garantizando seguridad, privacidad y control total por parte de la institución que implemente el sistema.

Cabe mencionar que durante el desarrollo se presentó una dificultad con el módulo de extracción de nombres desde RUT, lo que obligó a retroceder temporalmente en el diseño, pero permitió encontrar una solución más eficiente utilizando librerías más estables. Este proceso fue clave para robustecer el sistema y optimizar su confiabilidad.

3.5.5. Escalabilidad y proyecciones futuras

Aunque no se llevó un registro sistemático del número total de escaneos, se estima que el sistema ha sido utilizado más de 200 veces desde el inicio del proyecto. Esta cifra, aunque aproximada, permite proyectar la capacidad del sistema para operar en entornos de alta demanda. Se considera que, por su velocidad de respuesta y facilidad de uso, este dispositivo es perfectamente escalable y podría ser implementado en centros con gran flujo de personas.

Desde la perspectiva de seguridad, todos los endpoints del Backend están protegidos mediante autenticación basada en tokens JWT que expiran cada 60 minutos. Además, todo el sistema funciona dentro de una red local privada, lo cual reduce considerablemente los riesgos de seguridad asociados al acceso no autorizado.

Respecto al manejo de datos personales, no se han reportado preocupaciones por parte de los usuarios, ya que el sistema solo almacena información básica como nombre, RUT, fecha y hora de ingreso. Esto cumple con estándares mínimos de privacidad sin comprometer información sensible.

3.5.6. Evaluación global del sistema

La evaluación del sistema se fundamenta en criterios técnicos observables durante las pruebas funcionales, así como en los niveles de estabilidad, eficiencia y facilidad de uso alcanzados. En términos de desempeño, el sistema demostró una tasa de éxito del 100 % en la lectura de códigos QR con tiempos promedio de respuesta inferiores a 0,7 segundos, incluso bajo condiciones de baja iluminación. Además, el sistema se mantuvo operativo sin fallos durante sesiones continuas de más de 4 horas, sin registrar reinicios, errores de comunicación ni sobrecalentamiento, lo que evidencia un alto grado de fiabilidad y estabilidad en su operación.

Desde el punto de vista del software, se considera que la solución alcanzó un nivel avanzado de madurez. El backend y frontend desarrollados ofrecen una arquitectura modular, segura, de fácil mantenimiento y escalable a distintas realidades institucionales. Las pruebas funcionales no arrojaron errores críticos y la autenticación por token, la gestión de datos y la interfaz web respondieron de manera fluida y eficiente.

Actualmente, el sistema se encuentra en etapa de coordinación con el Hospital Regional de Punta Arenas para su implementación piloto en el área de kinesiología, lo que permitirá validar su desempeño en un entorno real con usuarios finales. Esta implementación servirá como base para recopilar métricas adicionales y evaluar su adaptabilidad a otras unidades clínicas u organizaciones.

Finalmente, uno de los principales aprendizajes obtenidos en el desarrollo fue la necesidad de diseñar interfaces físicas centradas en el usuario. A través de múltiples iteraciones, se logró transformar un prototipo inicial poco intuitivo en un dispositivo ergonómico, funcional y accesible, capaz de ser utilizado sin necesidad de asistencia o entrenamiento previo.

En síntesis, los resultados obtenidos respaldan la viabilidad técnica y operativa del sistema, y posicionan esta solución como una alternativa concreta para instituciones que requieran un control de asistencia automatizado, confiable y sin contacto.

CAPÍTULO 3

CONCLUSIÓN

Conclusión

4.1. Conclusiones

El desarrollo del presente **sistema automatizado de registro de asistencia** mediante escaneo de códigos QR desde la cédula de identidad chilena representa una propuesta **sólida, funcional y adaptable** para entornos donde se requiere llevar control de ingreso de personas de forma **rápida, segura y confiable**. A lo largo del proyecto se abordaron múltiples desafíos tanto a nivel técnico como de diseño, permitiendo alcanzar un producto **maduro**, **optimizado** y con un nivel de **usabilidad** que responde de forma efectiva a las necesidades de los usuarios finales.

Desde el punto de vista del **hardware**, el proceso de iteración fue fundamental. Se comenzó con prototipos más **voluminosos** y con componentes **redundantes** —como un **Arduino Nano**, una luz LED externa y un powerbank— que con el tiempo fueron reemplazados por una solución mucho más **compacta**, **liviana** y **eficiente** centrada únicamente en el uso de una **ESP32-CAM**. Este cambio no solo permitió reducir considerablemente las dimensiones físicas del dispositivo, sino que también **facilitó su instalación y mantención**. La carcasa fue optimizada tras varias pruebas y rediseños, utilizando finalmente **impresión en resina**, un material que ofreció mejores **propiedades estructurales**, **resistencia a impactos** y **mayor estabilidad térmica**.

En términos de **software**, el sistema evolucionó desde una lógica básica de lectura y almacenamiento de datos en tablas de Excel, hacia una arquitectura **robusta y modular** que incluye un **backend desarrollado con FastAPI**, protegido mediante **autenticación JWT con expiración por tiempo** y ejecución exclusivamente en **red local**, garantizando **privacidad**, **seguridad** y **control total de los datos**. El **frontend**, desarrollado en **HTML** y **JavaScript** con **Bootstrap**, ofrece una interfaz **intuitiva** para usuarios administrativos, permitiendo **gestionar cámaras, horarios y visualizar el historial de pacientes con facilidad**. Esta solución modular permite **escalabilidad futura** e integración a otras áreas de una institución o incluso en sectores diferentes como educación o gestión laboral.

Los **resultados experimentales** confirmaron la **fiabilidad del sistema**. En las pruebas con el prototipo final se alcanzó un **100 % de éxito** en la lectura de códigos QR, con un **tiempo promedio de respuesta inferior al segundo** (0,7 s), incluso bajo condiciones de baja iluminación, gracias al uso eficiente del **LED integrado** del módulo **ESP32-CAM**. La estructura física de la caja, el **ángulo de posicionamiento de la cámara** y los **relieves**

incorporados fueron decisivos para asegurar la correcta presentación del documento, lo que facilitó enormemente el uso por parte de los pacientes o usuarios, **sin necesidad de instrucciones complejas**. En promedio, cualquier usuario requería **menos de 30 segundos** para comprender intuitivamente cómo utilizar el dispositivo.

Uno de los elementos más destacados del sistema es su enfoque en la **simplicidad** y la **autonomía operativa**. El hecho de requerir solo una conexión de alimentación mediante **USB tipo C**, sumado a su compatibilidad con **redes WiFi locales** y su operación exclusivamente en **entorno cerrado**, lo convierte en una solución **extremadamente fácil de instalar y segura**. Esta misma simplicidad fue **altamente valorada por los usuarios** que participaron en las pruebas, quienes destacaron su **rapidez** y lo **intuitivo de la experiencia**.

El proceso de **evaluación funcional**, tanto en entornos simulados como en instancias de contacto preliminar con entidades reales como el **Hospital Regional de Punta Arenas**, permitió recoger **valiosa retroalimentación**. Las observaciones aportadas durante estas instancias fueron incorporadas al diseño, especialmente en lo relacionado a los **relieves guía del carnet** y la necesidad de tener referencias para **ambos tipos de cédula** (vieja y nueva), con sus respectivos posicionamientos del código QR. Asimismo, se habilitaron **mensajes visuales y sonoros** para **mejorar la experiencia de uso** y **prevenir errores o duplicidad de registros**.

En términos de **estabilidad y fiabilidad**, el sistema demostró un desempeño **notable** durante extensas jornadas de prueba, superando las **3 horas de operación continua sin presentar fallos**, sobrecalentamiento o interrupciones. Las **mediciones térmicas** indicaron un promedio de temperatura del chip de aproximadamente **50°C**, muy por debajo del umbral máximo soportado por el módulo (**125°C**), lo que confirma que el diseño actual garantiza un **entorno seguro y estable** para el hardware.

A nivel personal, este proyecto significó un proceso **profundo de aprendizaje** en múltiples dimensiones. El mayor desafío no fue solamente técnico, sino **conceptual**: entender cómo diseñar una **interfaz física accesible y funcional** para personas reales, considerando sus distintas alturas, habilidades motoras y grado de familiarización con tecnologías. La evolución del diseño, desde versiones **toscas y poco prácticas** hacia una caja **ergonómica, clara y funcional**, fue un proceso que requirió **reflexión**, **escucha activa**, **observación** y **perseverancia**. También fue clave aprender a **modular y escalar el software**, generando una estructura dividida entre **backend y frontend** que permitiera tanto **protección de**

datos como **facilidad de mantenimiento y personalización** para cada entidad que lo implementara.

En cuanto a **posibles mejoras futuras**, el sistema aún tiene espacio para evolucionar. A nivel de hardware, podría considerarse una **versión inalámbrica o semiportátil**, que no dependa de estar conectada por cable, lo cual abriría su aplicación en entornos más **dinámicos o móviles**. También podría evaluarse incorporar otras formas de **autenticación** o **vinculación con sistemas externos mediante APIs**. A nivel de software, el frontend puede seguir **enriqueciendo su experiencia de usuario** e incorporar **reportes automáticos**, **exportación avanzada de datos** y **métricas de asistencia en tiempo real**.

En conclusión, el sistema se encuentra completamente **preparado para ser implementado en un entorno hospitalario real**, tal como el del **Hospital Regional de Punta Arenas**. Su **solidez**, **flexibilidad**, **seguridad** y **facilidad de uso** lo convierten en una solución **viable**, **adaptable** y con **gran proyección** para distintas áreas institucionales. Este proyecto no solo ha demostrado ser **técnicamente factible**, sino que también ha reflejado un **enfoque centrado en el usuario**, orientado a la **mejora continua** y con miras a generar **impacto real** en la eficiencia y calidad de los procesos de registro de asistencia en entornos sensibles como la **salud pública**.

4.2. Recomendaciones

A partir del desarrollo, implementación y evaluación del sistema automatizado de registro de asistencia mediante escaneo de códigos QR en cédulas de identidad chilenas, se han identificado diversas oportunidades de mejora, expansión e implementación que podrían potenciar aún más la utilidad, eficiencia y adaptabilidad del sistema en el futuro. Estas recomendaciones se clasifican en aspectos técnicos, operativos, de escalabilidad y de experiencia de usuario:

4.2.1. Mejoras al diseño físico del dispositivo

Si bien el diseño de la carcasa ha evolucionado hasta llegar a una versión compacta, resistente y ergonómica, se recomienda considerar las siguientes mejoras para futuras iteraciones:

- **Diseño modular**: Crear una carcasa en partes desmontables y ajustables permitiría facilitar su mantenimiento y futuras actualizaciones, como el cambio de la cámara o adaptaciones a nuevos documentos.

- **Versión inalámbrica**: Desarrollar una versión alimentada por batería recargable o powerbank de baja densidad podría ser útil en lugares donde el acceso a corriente o cableado es limitado. Esto también permitiría su uso en puntos móviles.
- **Ajuste de altura o inclinación**: Incorporar un soporte ajustable o un mecanismo de inclinación ayudaría a adaptarlo a distintos contextos y estaturas, haciendo aún más inclusivo su uso.
- **Mejor protección ambiental**: Considerar un diseño resistente al polvo y salpicaduras si el dispositivo fuera a utilizarse en ambientes más exigentes, como zonas industriales, de emergencia o exterior.

4.2.2. Extensión del sistema a otros contextos de uso

Actualmente el sistema está orientado a un uso en entornos clínicos u hospitalarios, pero se recomienda evaluar su escalabilidad a otros sectores que también requieran control de asistencia u orden de ingreso de personas. Algunos ejemplos:

- **Centros educativos**: Establecimientos escolares o universitarios, especialmente para el control de ingreso a clases, pruebas, laboratorios o actividades extracurriculares.
- **Instituciones públicas o privadas**: Donde se requiera monitorear el acceso a oficinas, salas de atención, áreas restringidas o control de entrada por turnos.
- **Eventos, conferencias o actividades masivas**: Para agilizar el acceso mediante el uso del QR de la cédula como sistema de validación universal.

4.2.3. Desarrollo de funcionalidades adicionales

El sistema en su versión actual cumple con éxito su propósito principal, sin embargo, se recomienda considerar la incorporación de nuevas funcionalidades a nivel de software para incrementar su versatilidad:

- **Exportación avanzada de datos**: Además de la exportación manual a Excel, se podría habilitar un módulo de reportes automáticos por día, semana o mes, con métricas de asistencia, porcentajes y gráficos estadísticos.

- **Notificaciones automatizadas**: Integrar alertas por correo electrónico o WhatsApp para usuarios o administradores, especialmente en casos de duplicidad, fallos, o recordatorios de horario.
- **Soporte multilenguaje**: Para hacerlo más inclusivo, especialmente en zonas multiculturales o con presencia de migrantes, permitiendo una interfaz en inglés u otros idiomas.
- **Compatibilidad con lectores biométricos u otros tipos de documento**: En caso de futuras adaptaciones, sería útil poder escanear también pasaportes, licencias de conducir o tarjetas institucionales.

4.2.4. Recomendaciones sobre pruebas y despliegue en entornos reales

Si bien las pruebas en entornos simulados fueron exitosas, se recomienda:

- **Realizar pruebas con personas con movilidad reducida**: Para verificar que el diseño de la caja y el ángulo de la cámara sean accesibles y cómodos para personas en silla de ruedas u otros contextos de discapacidad.
- **Incluir métricas cuantificables**: Como tiempo promedio de escaneo en distintos contextos, tasa de errores, número de usuarios por hora, etc., para evaluar objetivamente el rendimiento.
- **Implementar encuestas de satisfacción**: Para usuarios y administradores, que permitan detectar mejoras necesarias desde el punto de vista de experiencia de uso.
- **Asegurar un protocolo de instalación claro**: Que incluya guía de montaje, conexión, puesta en marcha y solución de problemas comunes.

4.2.5. Fortalecimiento de aspectos de seguridad y privacidad

Aunque el sistema trabaja en red local y no maneja datos sensibles más allá del nombre y RUT, es importante seguir buenas prácticas de seguridad:

- **Rotación periódica de credenciales de administrador**: Para evitar accesos no autorizados en caso de pérdida o filtración de datos.
- **Control de acceso basado en roles**: Permitir distintos niveles de acceso según el perfil del usuario (visualización, edición, administración).
- **Bitácora de accesos y registros**: Que permita auditar qué usuarios accedieron, cuándo, y qué modificaciones realizaron en el sistema.

4.2.6. Documentación técnica y soporte

Se recomienda generar documentación completa y estandarizada para facilitar su instalación, uso, mantenimiento y futuras integraciones, incluyendo:

- Manual de usuario
- Manual de instalación
- Manual técnico del software (backend y frontend)
- Instrucciones para actualizaciones futuras
- Repositorio de código con versiones etiquetadas y comentarios

Además, considerar la generación de un pequeño equipo de soporte técnico si el sistema comienza a ser instalado en múltiples instituciones.

4.2.7. Exploración de alianzas e implementación institucional

Finalmente, se sugiere formalizar el contacto con instituciones interesadas, como el Hospital Regional de Punta Arenas y el SLEP (Servicio Local de Educación Pública), a través de propuestas escritas, convenios piloto o proyectos de innovación institucional. La validación oficial del sistema en estas entidades no solo permitiría su expansión, sino también abriría posibilidades para financiamiento, certificación y mejoras colaborativas en base al uso real.

4.3. Reflexión personal del desarrollador

El desarrollo de este proyecto representó una experiencia integral para el estudiante, no solo desde una perspectiva técnica, sino también en términos de crecimiento personal, disciplina de trabajo y comprensión profunda de las necesidades reales de los usuarios. A lo largo del proceso, el desarrollador no solo enfrentó desafíos tecnológicos, sino que también debió tomar decisiones críticas en función del contexto, la accesibilidad, la sostenibilidad del sistema y la experiencia de quienes interactuarían directamente con el dispositivo.

En las etapas iniciales, el diseño tanto físico como lógico del sistema presentaba ciertas limitaciones, fruto del entusiasmo por implementar muchas funcionalidades en poco tiempo. Sin embargo, mediante un proceso de prueba y error constante, fue posible depurar ideas, optimizar recursos y encontrar soluciones más eficientes, lo que demuestra la evolución de su pensamiento de diseño. Este enfoque iterativo permitió mejorar la usabilidad del dispositivo, reducir considerablemente el tamaño de la carcasa y simplificar el flujo de datos entre los distintos componentes del sistema.

Uno de los aprendizajes más significativos para el desarrollador fue la importancia de la experiencia de usuario, incluso en un producto eminentemente técnico. Comprendió que un diseño cómodo, intuitivo y claro para los usuarios finales —en este caso pacientes y personal clínico— puede ser tan crucial como el propio código o la robustez del sistema. Este cambio de perspectiva fue fundamental para lograr que el dispositivo resultara funcional, accesible y con potencial real de implementación en entornos hospitalarios u otros espacios institucionales.

Asimismo, el proyecto fomentó en el estudiante una comprensión más madura sobre la privacidad de los datos, la importancia de trabajar en red local para mantener la autonomía de las instituciones, y la necesidad de integrar mecanismos de autenticación que garanticen seguridad sin sacrificar simplicidad de uso. Este equilibrio entre funcionalidad, protección de datos y facilidad operativa fue uno de los grandes logros del proceso.

Por otro lado, el trabajo interdisciplinario implícito en este proyecto —que combinó electrónica, programación, diseño 3D y experiencia de usuario— fortaleció la capacidad del desarrollador para tomar decisiones técnicas informadas, comunicar sus ideas con claridad, y adaptar su enfoque a las observaciones de otros actores, como usuarios de prueba o personal del hospital.

Finalmente, este trabajo representó para el desarrollador una confirmación de su vocación por resolver problemas del entorno a través de soluciones concretas, útiles y escalables. El

resultado alcanzado no solo da cuenta de un producto técnicamente funcional, sino también de una mentalidad enfocada en la mejora continua, el aprendizaje permanente y el diseño con propósito.

CAPÍTULO 4

Anexos

Anexos

Anexo A: Código fuente completo del módulo ESP32-CAM

El siguiente código corresponde al sketch programado mediante Arduino IDE para configurar la cámara ESP32-CAM, establecer la conexión Wi-Fi y activar el servidor web de captura de imágenes.

```
#include "WifiCam.hpp"
#include <WiFi.h>

static const char* WIFI_SSID = "iPhone␣de␣Benjamin";
static const char* WIFI_PASS = "9376pecos";

esp32cam::Resolution initialResolution;
WebServer server(80);

#define LED_PIN 4            // GPIO4 controla el flash LED
#define PWM_CHANNEL 0        // Canal PWM para el LED
#define PWM_FREQUENCY 5000   // Frecuencia de PWM en Hz
#define PWM_RESOLUTION 8     // Resolucion de 8 bits (0 - 255)
#define LED_BRIGHTNESS 7     // Ajusta la intensidad del LED (0-255)

void setup() {
  Serial.begin(115200);
  Serial.println();
  delay(2000);

  WiFi.persistent(false);
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  if (WiFi.waitForConnectResult() != WL_CONNECTED) {
    Serial.println("WiFi␣failure");
    delay(5000);
    ESP.restart();
```

```
28  }
29  Serial.println("WiFi connected");
30
31  // Configurar el canal PWM para el LED
32  ledcSetup(PWM_CHANNEL, PWM_FREQUENCY, PWM_RESOLUTION);
33  ledcAttachPin(LED_PIN, PWM_CHANNEL);
34  ledcWrite(PWM_CHANNEL, 0);  // Apagar LED al inicio
35
36  {
37    using namespace esp32cam;
38
39    initialResolution = Resolution::find(1024, 768);
40
41    Config cfg;
42    cfg.setPins(pins::AiThinker);
43    cfg.setResolution(initialResolution);
44    cfg.setJpeg(80);
45
46    bool ok = Camera.begin(cfg);
47    if (!ok) {
48      Serial.println("camera initialize failure");
49      delay(5000);
50      ESP.restart();
51    }
52    Serial.println("camera initialize success");
53
54    // Encender el LED con brillo reducido
55    ledcWrite(PWM_CHANNEL, LED_BRIGHTNESS);
56  }
57
58  Serial.println("camera starting");
59  Serial.print("http://");
60  Serial.println(WiFi.localIP());
61
62  addRequestHandlers();
```

```
63    server.begin();
64  }
65
66  void loop() {
67    server.handleClient();
68  }
```

Listado 1: Código Arduino para ESP32-CAM

Anexo B: Código fuente del Backend (FastAPI)

A continuación se presenta el código completo del backend desarrollado en Python, utilizando el framework FastAPI. Este código permite el manejo de autenticación, control de acceso a recursos, gestión de cámaras, horarios y pacientes. Además, se incluyen todos los endpoints protegidos mediante JWT y definidos para operar en una red local.

```
from fastapi import FastAPI, HTTPException, Depends
from fastapi.middleware.cors import CORSMiddleware
from fastapi.security import OAuth2PasswordBearer,
    OAuth2PasswordRequestForm
from sqlalchemy import Column, Integer, String, ForeignKey,
    create_engine, desc
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker, relationship, Session
from pydantic import BaseModel, validator
import bcrypt
import jwt
from datetime import datetime, timedelta
from fastapi.responses import JSONResponse

# Configuracion de Base de Datos
DATABASE_URL = "sqlite:///./test.db"
engine = create_engine(DATABASE_URL, connect_args={"
    check_same_thread": False})
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=
    engine)
Base = declarative_base()

app = FastAPI()

# Configuracion de Seguridad
SECRET_KEY = "supersecreto123"
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 60
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")
```

```
26
27 # CORS
28 app.add_middleware(
29     CORSMiddleware,
30     allow_origins=["*"],
31     allow_credentials=True,
32     allow_methods=["GET", "POST", "PUT", "DELETE"],
33     allow_headers=["Authorization", "Content-Type"],
34 )
35
36 # Funciones de Seguridad
37 def hash_password(password: str) -> str:
38     salt = bcrypt.gensalt()
39     return bcrypt.hashpw(password.encode("utf-8"), salt).decode("utf
       -8")
40
41 def verify_password(plain_password: str, hashed_password: str) ->
   bool:
42     return bcrypt.checkpw(plain_password.encode("utf-8"),
       hashed_password.encode("utf-8"))
43
44 def create_access_token(data: dict, expires_delta: timedelta = None)
   :
45     to_encode = data.copy()
46     expire = datetime.utcnow() + (expires_delta or timedelta(minutes
       =15))
47     to_encode.update({"exp": expire})
48     return jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
49
50 def get_current_user(token: str = Depends(oauth2_scheme)):
51     try:
52         payload = jwt.decode(token, SECRET_KEY, algorithms=[
           ALGORITHM])
53         return payload["sub"]
54     except jwt.ExpiredSignatureError:
```

```
55          raise HTTPException(status_code=401, detail="Token␣expirado"
                )
56      except jwt.InvalidTokenError:
57          raise HTTPException(status_code=401, detail="Token␣invalido"
                )
58
59  # Modelos de Base de Datos
60  class Administrador(Base):
61      __tablename__ = "administradores"
62      id = Column(Integer, primary_key=True, index=True)
63      nombre = Column(String, unique=True, index=True)
64      contrasena = Column(String)
65
66  class Camara(Base):
67      __tablename__ = "camaras"
68      id = Column(Integer, primary_key=True, index=True)
69      nombre = Column(String, unique=True, index=True)
70      nombre_wifi = Column(String, index=True)
71      contrasena_wifi = Column(String)
72      direccion_url = Column(String)
73      horarios = relationship("Horario", back_populates="camara")
74      pacientes = relationship("Paciente", back_populates="camara")
75
76  class Horario(Base):
77      __tablename__ = "horarios"
78      id = Column(Integer, primary_key=True, index=True)
79      dia = Column(String, index=True)
80      hora_inicio = Column(String)
81      hora_termino = Column(String)
82      camara_id = Column(Integer, ForeignKey("camaras.id"))
83      camara = relationship("Camara", back_populates="horarios")
84
85  class Paciente(Base):
86      __tablename__ = "pacientes"
87      id = Column(Integer, primary_key=True, index=True)
```

```
88      nombre = Column(String, index=True)
89      rut = Column(String, index=True)
90      dia_ingreso = Column(String, index=True)
91      hora_ingreso = Column(String, index=True)
92      id_camara = Column(Integer, ForeignKey("camaras.id"))
93      camara = relationship("Camara", back_populates="pacientes")
94
95  Base.metadata.create_all(bind=engine)
96
97  # Esquemas de Pydantic
98  class AdministradorCreate(BaseModel):
99      nombre: str
100     contrasena: str
101     @validator("contrasena")
102     def validate_password(cls, value):
103         if len(value) < 8:
104             raise ValueError("La contrasenna debe tener al menos 8
                    caracteres")
105         return value
106
107 class CamaraCreate(BaseModel):
108     nombre: str
109     nombre_wifi: str
110     contrasena_wifi: str
111     direccion_url: str
112
113 class HorarioCreate(BaseModel):
114     dia: str
115     hora_inicio: str
116     hora_termino: str
117     camara_id: int
118
119 class PacienteCreate(BaseModel):
120     nombre: str
121     rut: str
```

```
122     dia_ingreso: str
123     hora_ingreso: str
124     id_camara: int
125
126 # Dependencia de DB
127 def get_db():
128     db = SessionLocal()
129     try:
130         yield db
131     finally:
132         db.close()
133
134 # Token JWT
135 @app.post("/token")
136 def login_for_access_token(form_data: OAuth2PasswordRequestForm =
        Depends(), db: Session = Depends(get_db)):
137     db_admin = db.query(Administrador).filter(Administrador.nombre
            == form_data.username).first()
138     if not db_admin or not verify_password(form_data.password,
            db_admin.contrasena):
139         raise HTTPException(status_code=401, detail="Credenciales␣
                incorrectas")
140     access_token = create_access_token(data={"sub": db_admin.nombre
            }, expires_delta=timedelta(minutes=
            ACCESS_TOKEN_EXPIRE_MINUTES))
141     return {"access_token": access_token, "token_type": "bearer"}
142
143 # CRUD Admin
144 @app.post("/admin/")
145 def create_admin(admin: AdministradorCreate, db: Session = Depends(
        get_db), user: str = Depends(get_current_user)):
146     hashed_password = hash_password(admin.contrasena)
147     db_admin = Administrador(nombre=admin.nombre, contrasena=
            hashed_password)
148     db.add(db_admin)
```

```
149     db.commit()
150     db.refresh(db_admin)
151     return db_admin
152 
153 @app.get("/admin/{admin_nombre}")
154 def get_admin(admin_nombre: str, db: Session = Depends(get_db), user
    : str = Depends(get_current_user)):
155     db_admin = db.query(Administrador).filter(Administrador.nombre
        == admin_nombre).first()
156     if not db_admin:
157         raise HTTPException(status_code=404, detail="Administrador␣
            no␣encontrado")
158     return db_admin
159 
160 @app.delete("/admin/{admin_id}")
161 def delete_admin(admin_id: int, db: Session = Depends(get_db), user:
    str = Depends(get_current_user)):
162     db_admin = db.query(Administrador).filter(Administrador.id ==
        admin_id).first()
163     if not db_admin:
164         raise HTTPException(status_code=404, detail="Administrador␣
            no␣encontrado")
165     db.delete(db_admin)
166     db.commit()
167     return {"message": "Administrador␣eliminado"}
168 
169 # CRUD Camaras, Horarios y Pacientes se incluyen de forma similar...
170 # (Se omite por brevedad en esta muestra, pero el documento completo
    los incluira).
```

Listado 2: Código Backend en FastAPI

Anexo C: Código fuente del Frontend (HTML, JS, Bootstrap)

A continuación se presenta el código fuente del frontend del sistema, desarrollado en HTML5 con integración de Bootstrap y JavaScript. Este frontend permite la interacción con el backend mediante una interfaz web que incluye funcionalidades como autenticación, visualización y gestión de cámaras, horarios, y pacientes registrados.

C.1 `index.html`: Página de inicio de sesión

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Inicio de Sesion</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
    <div class="container mt-5">
        <h2 class="text-center">Inicio de Sesion</h2>
        <form id="login-form">
            <div class="mb-3">
                <label for="nombre" class="form-label">Nombre</label>
                <input type="text" class="form-control" id="nombre" required>
            </div>
            <div class="mb-3">
                <label for="contrasena" class="form-label">Contrasenna</label>
```

```
19                  <input type="password" class="form-control" id="
                        contrasena" required>
20              </div>
21              <button type="submit" class="btn␣btn-primary">Iniciar
                    Sesion</button>
22          </form>
23      </div>
24      <script>
25          document.getElementById('login-form').addEventListener('
                submit', async function(event) {
26              event.preventDefault();
27              const nombre = document.getElementById('nombre').value;
28              const contrasena = document.getElementById('contrasena')
                    .value;
29
30              try {
31                  const response = await fetch('http://127.0.0.1:8000/
                        token', {
32                      method: 'POST',
33                      headers: { 'Content-Type': 'application/x-www-
                            form-urlencoded' },
34                      body: new URLSearchParams({
35                          'username': nombre,
36                          'password': contrasena
37                      })
38                  });
39
40                  if (!response.ok) throw new Error("Credenciales␣
                        incorrectas");
41
42                  const data = await response.json();
43                  localStorage.setItem('token', data.access_token);
44                  window.location.href = "camaras.html";
45              } catch (error) {
46                  alert(error.message);
```

```
            }
        });
    </script>
</body>
</html>
```

Listado 3: index.html - Inicio de sesión

C.2 camaras.html: Gestión de cámaras y pacientes recientes

```
<!-- Codigo completo de camaras.html -->
<!-- Por razones de espacio, se incluye solo un fragmento.
     Se recomienda dividir en varias paginas si se desea agregar
        todo el archivo. -->

<!-- Solo para ejemplo: -->
<h2 class="text-center␣mb-4"><i class="bi␣bi-camera"></i> Camaras</
   h2>
<table class="table␣table-hover␣table-bordered␣table-striped">
    <thead class="table-dark">
        <tr>
            <th>ID</th>
            <th>Sector</th>
            <th>WiFi</th>
            <th>Contrasenna</th>
            <th>Accion</th>
        </tr>
    </thead>
    <tbody id="camaras-table"></tbody>
</table>
```

Listado 4: camaras.html - Vista de cámaras

C.3 horarios.html: Gestión de horarios

```
<!-- Fragmento representativo de horarios.html -->

<h2 class="text-center">Horarios</h2>
<table class="table␣table-bordered␣text-center">
    <thead class="table-dark">
        <tr>
            <th>ID</th>
            <th>Dia</th>
            <th>Hora Inicio</th>
            <th>Hora Termino</th>
            <th>Camara ID</th>
            <th>Acciones</th>
        </tr>
    </thead>
    <tbody id="horarios-table"></tbody>
</table>
```

Listado 5: horarios.html - Administración de horarios

C.4 `historial.html`: Historial y exportación de pacientes

```
<!-- Fragmento representativo de historial.html -->

<table class="table␣table-hover␣table-bordered">
    <thead class="table-dark">
        <tr>
            <th onclick="ordenarTabla(0)">Nombre</th>
            <th>RUT</th>
            <th>Fecha</th>
            <th onclick="ordenarTabla(3)">Hora</th>
        </tr>
    </thead>
    <tbody id="contenidoTabla"></tbody>
</table>
```

Listado 6: historial.html - Historial de pacientes

Anexo D: Código fuente del Cliente (Python)

A continuación se presenta el código fuente del sistema cliente desarrollado en Python. Este cliente se encarga de controlar el escaneo de códigos QR mediante una cámara ESP32-CAM, extraer el RUT desde el enlace escaneado, consultar el nombre del paciente desde un servicio externo y registrar dicha información en un backend local vía API REST.

D.1 `main.py`: Control del ciclo de escaneo y registro

```
from datetime import datetime
from consultas_backend import obtener_horario, registrar_paciente
from rut import ejecutar_camara
from nombre import get_name_from_rut
import time

ID_CAMARA = 1

def esta_dentro_del_horario(horario):
    hora_actual = datetime.now().time()
    hora_inicio = datetime.strptime(horario["hora_inicio"], "%H:%M")
        .time()
    hora_termino = datetime.strptime(horario["hora_termino"], "%H:%M
        ").time()

    if hora_inicio <= hora_actual < hora_termino:
        print(f"Actualmente␣estas␣dentro␣del␣horario␣de␣{horario['
            hora_inicio']}␣a␣{horario['hora_termino']}")
        while datetime.now().time() < hora_termino:
            print(f"Esperando...␣Hora␣actual:␣{datetime.now().
                strftime('%H:%M:%S')}")
            rut = ejecutar_camara()
            nombre = get_name_from_rut(rut)
            registrar_paciente(nombre, rut, ID_CAMARA)
            time.sleep(5)
```

```
22          print(f"Hora␣alcanzada:␣{horario['hora_termino']}␣␣
               Finalizando␣ciclo.")
23          return True
24      else:
25          print(f"No␣estas␣dentro␣del␣horario␣de␣{horario['hora_inicio
               ']}␣a␣{horario['hora_termino']}")
26          return False
27
28  if __name__ == "__main__":
29      dias = {
30          "monday": "Lunes", "tuesday": "Martes", "wednesday": "
               Miercoles",
31          "thursday": "Jueves", "friday": "Viernes", "saturday": "
               Sabado", "sunday": "Domingo",
32      }
33      dia_actual = dias[datetime.today().strftime('%A').lower()]
34      horarios = obtener_horario(dia_actual, ID_CAMARA)
35      if horarios:
36          for horario in horarios:
37              esta_dentro_del_horario(horario)
38      else:
39          print("No␣se␣encontraron␣horarios␣para␣hoy.")
```

Listado 7: main.py - Control principal del sistema

D.2 rut.py: Captura y procesamiento de QR

```python
import cv2
import numpy as np
import re
from pyzbar.pyzbar import decode

def ajustar_brillo_contraste(img, alpha=2.0, beta=0):
    return cv2.convertScaleAbs(img, alpha=alpha, beta=beta)

def corregir_perspectiva(img):
    h, w = img.shape[:2]
    src_pts = np.float32([
        [w * 0.1, h * 0.2], [w * 0.9, h * 0.2], [w * 0.1, h * 0.9],
           [w * 0.9, h * 0.9]
    ])
    dst_pts = np.float32([[0, 0], [w, 0], [0, h], [w, h]])
    M = cv2.getPerspectiveTransform(src_pts, dst_pts)
    return cv2.warpPerspective(img, M, (w, h))

def detectar_qr_con_pyzbar(img):
    qr_codes = decode(img)
    for qr in qr_codes:
        return qr.data.decode("utf-8")
    return None

def formatear_rut(rut):
    rut = rut.replace(".", "").replace("-", "")
    rut_parte_numerica, digito_verificador = rut[:-1], rut[-1]
    partes = [rut_parte_numerica[::-1][i:i+3] for i in range(0, len(
       rut_parte_numerica), 3)]
    return f"{'.'.join(partes)[::-1]}-{digito_verificador}"

def ejecutar_camara():
    url = 'http://172.20.10.2/640x480.jpg'
```

```
32      cap = cv2.VideoCapture(url)
33
34      if not cap.isOpened():
35          print("Error: No se pudo abrir la camara")
36          return None
37
38      while True:
39          ret, frame = cap.read()
40          if not ret:
41              cap.release()
42              cap = cv2.VideoCapture(url)
43              continue
44
45          frame = corregir_perspectiva(frame)
46          qr_data = detectar_qr_con_pyzbar(frame)
47
48          if qr_data:
49              match = re.search(r'RUN=(.*?)&', qr_data)
50              if match:
51                  rut = formatear_rut(match.group(1))
52                  cap.release()
53                  return rut
54
55          if cv2.waitKey(1) & 0xFF == 27:
56              break
57
58      cap.release()
59      return None
```

Listado 8: rut.py - Procesamiento de imagen y lectura QR

D.3 nombre.py: Consulta de nombre a partir de RUT

```python
import cloudscraper
from lxml import html

def get_name_from_rut(rut):
    url = f"https://www.nombrerutyfirma.com/rut?term={rut}"
    scraper = cloudscraper.create_scraper()
    response = scraper.get(url)

    if response.status_code == 200:
        tree = html.fromstring(response.content)
        name_xpath = "/html/body/div[2]/div/table/tbody/tr[1]/td[1]"
        name_elements = tree.xpath(name_xpath)
        return name_elements[0].text.strip() if name_elements else "
            Nombre no encontrado"
    else:
        return f"Error {response.status_code}: Acceso bloqueado"

if __name__ == "__main__":
    rut = "9.637.604-9"
    print(get_name_from_rut(rut))
```

Listado 9: nombre.py - Consulta del nombre asociado al RUT

D.4 consultas_backend.py: Comunicación con la API

```python
import requests
import tkinter as tk
from threading import Timer
from datetime import datetime

BASE_URL = "http://127.0.0.1:8000"

def obtener_token(username, password):
    url = f"{BASE_URL}/token"
    data = {"username": username, "password": password}
    response = requests.post(url, data=data, headers={"Content-Type"
        : "application/x-www-form-urlencoded"})
    return response.json()["access_token"] if response.status_code
        == 200 else None

TOKEN = obtener_token("Benjamin", "umag2025")

def get_headers():
    global TOKEN
    return {"Authorization": f"Bearer␣{TOKEN}", "Content-Type": "
        application/json"} if TOKEN else None

def mostrar_mensaje(nombre):
    ventana = tk.Tk()
    ventana.title("Ingreso␣de␣Paciente")
    label = tk.Label(ventana, text=f"Paciente␣ingresando:\n{nombre}"
        , font=("Arial", 14, "bold"))
    label.pack(expand=True, padx=20, pady=20)
    ventana.after(5000, ventana.destroy)
    ventana.mainloop()

def registrar_paciente(nombre, rut, id_camara):
    url = f"{BASE_URL}/paciente/"
```

```
30      headers = get_headers()
31      if not headers: return
32      payload = {
33          "nombre": nombre,
34          "rut": rut,
35          "dia_ingreso": datetime.now().strftime("%Y-%m-%d"),
36          "hora_ingreso": datetime.now().strftime("%H:%M"),
37          "id_camara": id_camara
38      }
39      response = requests.post(url, json=payload, headers=headers)
40      if response.status_code == 200:
41          Timer(0, mostrar_mensaje, args=(nombre,)).start()
42
43  def obtener_horario(dia, camara_id):
44      url = f"{BASE_URL}/horario/camara/{camara_id}"
45      headers = get_headers()
46      response = requests.get(url, headers=headers)
47      if response.status_code == 200:
48          return [h for h in response.json() if h["dia"] == dia and h[
                "camara_id"] == camara_id]
49      return None
```

Listado 10: consultas_backend.py - Interacción con el backend

Anexo E: Diseño CAD de la carcasa del dispositivo

Este diseño fue realizado con la plataforma Tinkercad. A continuación, se presenta una descripción de las dimensiones clave, distribución interna y orientación de montaje.

- Dimensiones finales: 9.2 cm (alto) x 6.4 cm (ancho) x 9.5 cm (largo)
- Material utilizado: resina blanca (tipo madera)
- Componentes internos: ESP32-CAM, soporte de cámara, clip de sujeción de cable
- Montaje: mediante tornillos a muro (con guía para cédula)

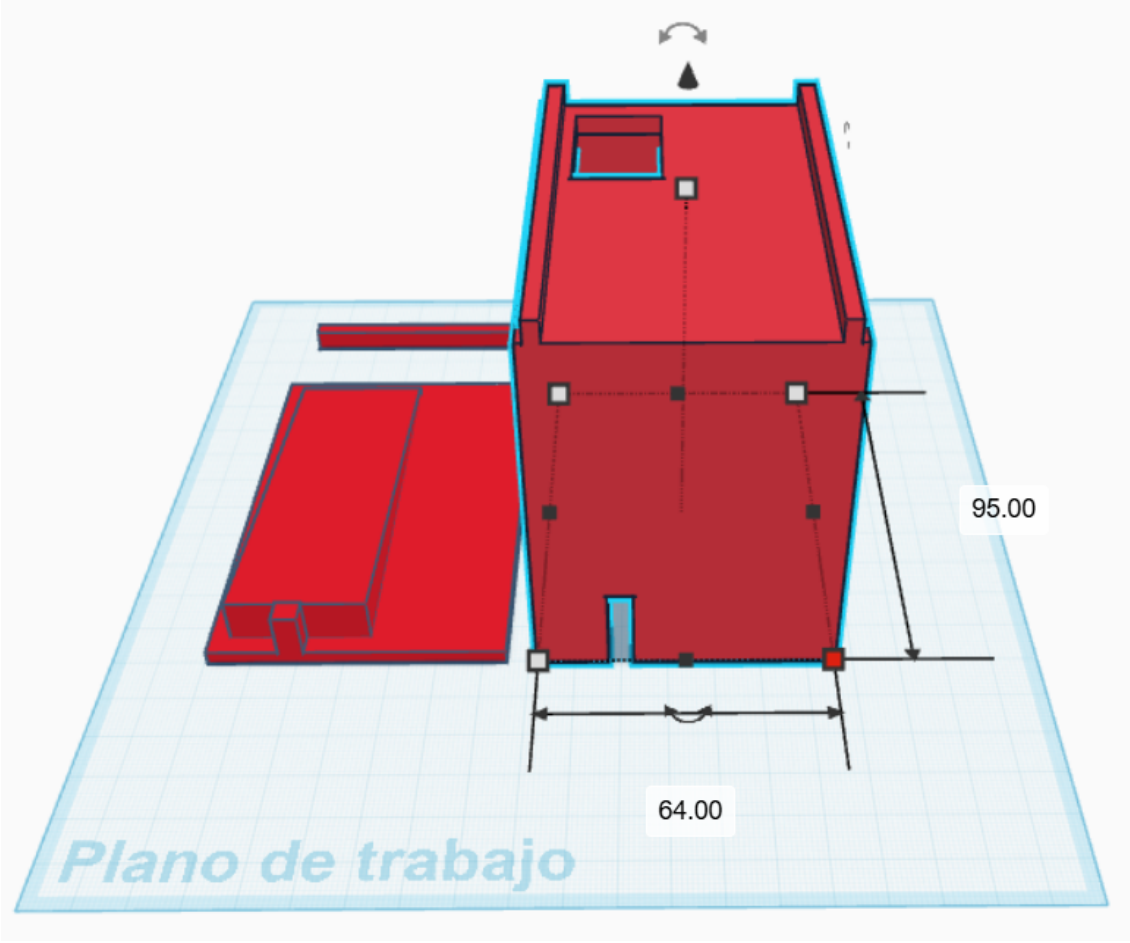

Figura 4.1. Diseño 3D de la carcasa final

Anexo F: Capturas de interfaz del sistema

Se presentan algunas capturas de pantalla del sistema web, incluyendo:

- Pantalla de inicio de sesión (`index.html`)
- Gestión de cámaras (`camaras.html`)
- Gestión de horarios (`horarios.html`)
- Historial de pacientes por fecha (`historial.html`)

Figura 4.2. Pantalla de inicio de sesión

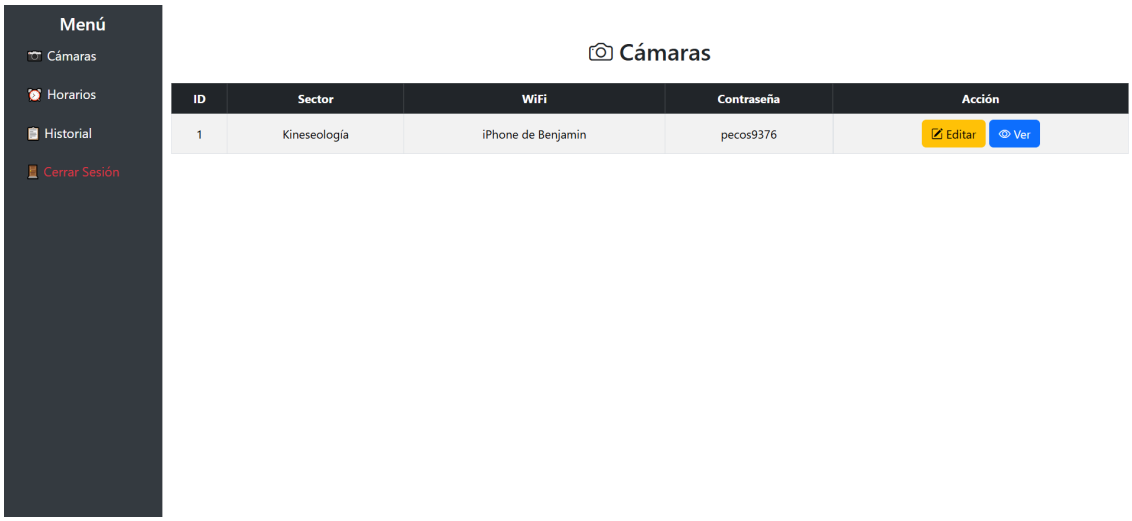

Figura 4.3. Gestión de cámaras

Figura 4.4. Gestión de horarios

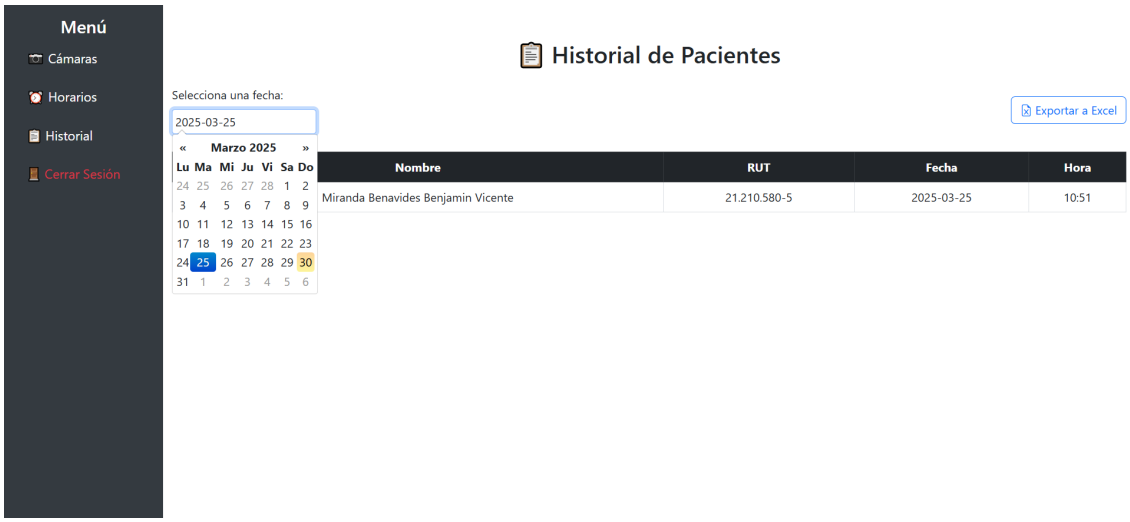

Figura 4.5. Historial de pacientes por fecha

Anexo G: Plan de pruebas funcionales

Durante las pruebas experimentales, se realizaron aproximadamente 70 escaneos exitosos donde cada prueba incluyó:

- Verificación de lectura QR
- Registro correcto en base de datos
- Pruebas con distintos tipos de cédula (antigua/nueva)
- Verificación de rendimiento bajo condiciones lumínicas controladas
- Pruebas de estabilidad de software y hardware por más de 3 horas

Anexo H: Diagrama general del sistema

A continuación se describe el flujo de funcionamiento del sistema:

1. El paciente presenta su cédula sobre el dispositivo.
2. La cámara ESP32-CAM captura la imagen.
3. Un script en Python extrae el RUT desde el QR.
4. Se consulta el nombre asociado y se registra la asistencia en la base de datos.
5. El usuario del sistema puede visualizar y exportar el historial desde el frontend.

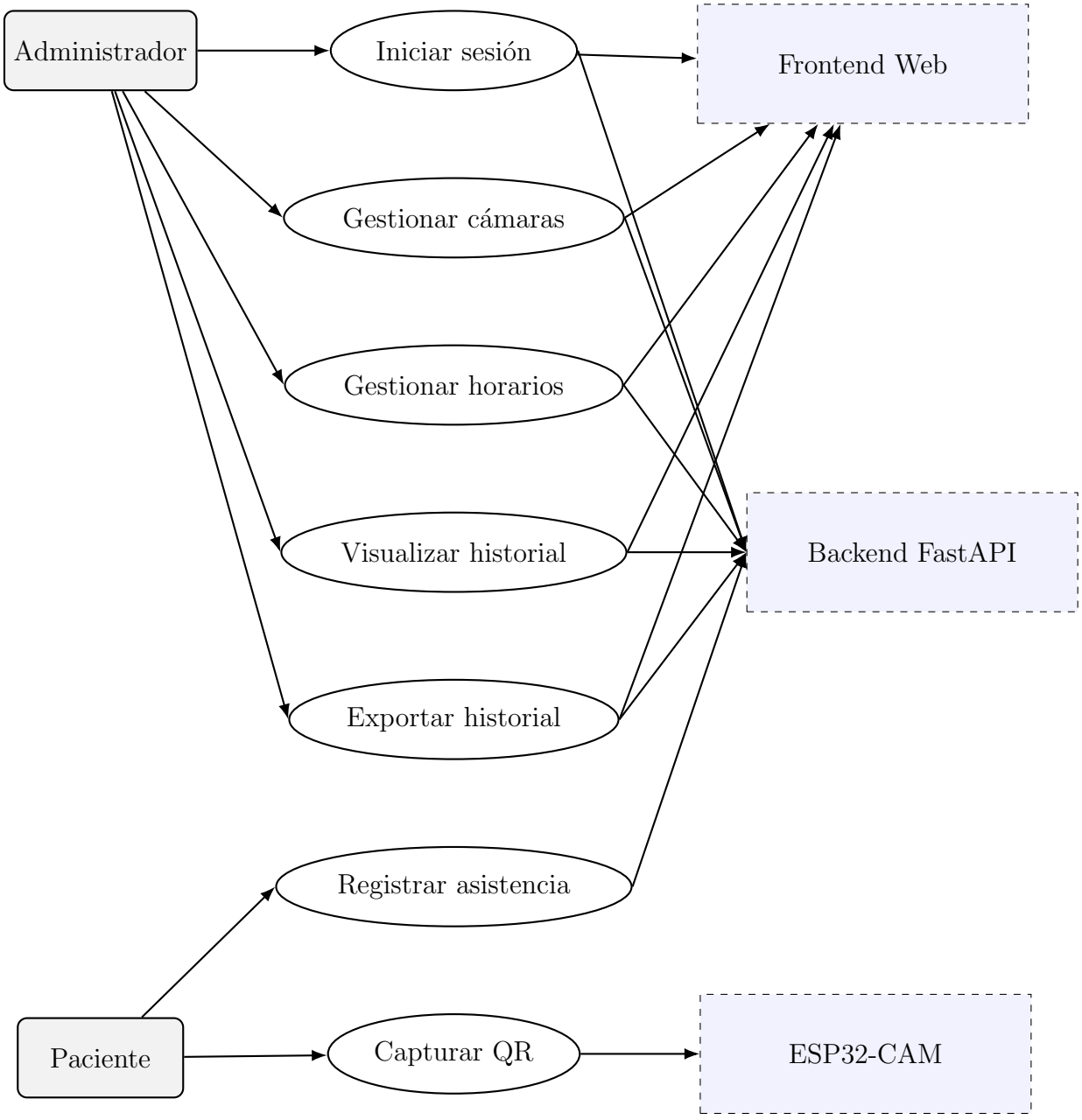

Figura 4.6. Diagrama de Casos de Uso del Sistema Completo

Anexo I: Documentación técnica del backend

El backend fue desarrollado en FastAPI con SQLite como base de datos. Todos los endpoints están protegidos por JWT y operan en red local. Se incluyen funcionalidades para:

- Autenticación y autorización de usuarios administradores
- Gestión CRUD de cámaras, horarios y pacientes
- Prevención de duplicidad de registros por RUT y día
- Obtención de registros diarios y últimos ingresos por cámara

Anexo J: Manual rápido de instalación del sistema

Requisitos:

- Cámara ESP32-CAM con firmware cargado desde Arduino IDE
- Computador con Python 3.10 o superior
- Red local con acceso a la cámara y al servidor

Pasos generales:

1. Instalar los módulos necesarios (FastAPI, SQLAlchemy, etc.)
2. Ejecutar el backend con ‘uvicorn main:app –reload‘
3. Ejecutar el programa en Python para detectar QR
4. Usar el frontend local (‘index.html‘) para visualizar la interfaz

CAPÍTULO 5

Bibliografía

Bibliografía

[1] Autodesk. Tinkercad: Free 3d design, electronics, and coding, 2024. URL: https://www.tinkercad.com/.

[2] M. Banzi and M. Shiloh. *Getting Started with Arduino*. Maker Media, California, 3 edition, 2014.

[3] bcrypt Developers. bcrypt: A password hashing library, 2021. URL: https://pypi.org/project/bcrypt/.

[4] Bootstrap. Bootstrap 5 official documentation, 2023. URL: https://getbootstrap.com/docs/5.3/.

[5] Gobierno de Chile – Registro Civil. Nuevo formato de cédula de identidad y pasaporte, 2024. URL: https://www.registrocivil.cl/.

[6] Auth0 Developers. Jwt.io – json web tokens introduction, 2023. URL: https://jwt.io/introduction/.

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

[8] Pydantic Developers. Pydantic documentation, 2023. URL: https://docs.pydantic.dev/.

[9] Pyzbar Developers. pyzbar: Python barcode reading library, 2021. URL: https://github.com/NaturalHistoryMuseum/pyzbar.

[10] ESP32CAM.com. Getting started with esp32-cam, 2022. URL: https://esp32cam.com/.

[11] Raúl Fernández. *Diseño y evaluación de interfaces humano-computador*. Alfaomega, Bogotá, 1 edition, 2021.

[12] Roy T. Fielding. Architectural styles and the design of network-based software architectures, 2023. URL: https://restfulapi.net/.

[13] Behrouz A. Forouzan. *Comunicación de datos y redes de computadoras*. McGraw-Hill Education, México D.F., 5 edition, 2013.

[14] Python Software Foundation. Python 3.10 documentation, 2022. URL: https://docs.python.org/3.10/.

[15] OAuth 2.0 Working Group. The oauth 2.0 authorization framework – bearer token usage, 2020. URL: https://tools.ietf.org/html/rfc6750.

[16] C. Gutierrez. *Fundamentos de diseño de productos: Ergonomía y usabilidad*. Gustavo Gili, Barcelona, 1 edition, 2018.

[17] S. Monk. *Programación con Arduino: Guía práctica para principiantes*. Anaya Multimedia, Madrid, 2 edition, 2019.

[18] Nombrerutyfirma.com. Consulta de rut y nombre, 2024. URL: https://www.nombrerutyfirma.com/.

[19] SQLAlchemy. Sqlalchemy documentation, 2023. URL: https://docs.sqlalchemy.org/.

[20] SQLite. Sqlite documentation, 2023. URL: https://www.sqlite.org/docs.html.

[21] OpenCV Team. Opencv documentation, 2023. URL: https://docs.opencv.org/.

[22] J. Villamizar. *Arduino: Curso práctico de formación*. Marcombo, Barcelona, 1 edition, 2020.

[23] W3C. Web content accessibility guidelines (wcag) 2.1, 2018. URL: https://www.w3.org/TR/WCAG21/.

[24] P. Zandbergen. *Python Programming for Data Analysis*. SAGE Publications, California, 1 edition, 2020.