MEMORIA TÉCNICA
RING DOOR

Sistemas Empotrados y De Tiempo Real
Grado de Ingeniería Informática
URJC, Vicálvaro.

Joaquín Fuentes López
Sergio Barrios Paz
Jorge Sánchez Aguilar

Índice

Apunte importante: Durante esta memoría verá texto en color verde, estos hacen referencia a archivos o funciones del repositorio, por ello es muy recomendable que que se tenga a mano el repositorio para ir viendo de manera detallada estas funciones conforme se avanza esta lectura.

1. Reparto de tareas

A lo largo del desarrollo del proyecto, todos los integrantes del grupo han colaborado activamente en las diferentes fases, incluyendo el diseño, implementación y documentación. Sin embargo, se ha producido una especialización natural en algunas áreas:

- **Jorge**: ha sido el principal responsable del desarrollo del servidor web (backend y frontend).

- **Sergio**: se ha centrado en el código arduino y la integración del hardware, especialmente en la comunicación entre Arduino y ESP32.

- **Joaquín**: encargado principalmente del diseño y montaje de la maqueta física y la conexión de **componentes hardware**.

2. Coste de los materiales

En la siguiente tabla se muestra el coste de los materiales que se compraron para el proyecto. El resto de materiales (Arduino, cables, buzzer, maqueta, protoboard, etc.) fueron proporcionados por la universidad y no suponen un coste adicional para el grupo.

Componente	Cantidad	Precio unitario (€)	Precio total (€)
ESP32-CAM AI-Thinker	1	7,99	7,99
Pantalla OLED SSD1306	1	1,59	1,59
Módulo NFC MFRC522	1	1,19	1,19
Sensor PIR HC-SR501	1	1,99	1,99

Módulo Lector de Huellas (R503)	1	6,89	6,89
Total Coste Directo	—	—	**19,65 €**

3. Pasos dados

Para el desarrollo del sistema se siguió una **metodología bottom-up**, construyendo el sistema desde los componentes más básicos hasta alcanzar la funcionalidad completa. A continuación, se describen los pasos principales realizados:

1. **Selección de la idea y definición de objetivos**
   El equipo propuso un sistema de control de acceso multifactor que combinara seguridad física con monitorización remota. Se eligió implementar autenticación mediante huella y NFC, junto con un sistema de alertas y gestión desde un servidor web.

2. **Montaje y prueba individual de componentes**
   Cada sensor y módulo (NFC, huellas, PIR, OLED, etc.) fue conectado y probado de forma individual en el Arduino para verificar su correcto funcionamiento.

3. **Desarrollo del firmware embebido**

   - Se comenzó por el código del **Arduino**, desarrollando la lógica de autenticación, lectura de sensores y control de actuadores.

   - Una vez que teníamos un código Arduino bastante estructurado, se programó la **ESP32-CAM** para gestionar la conectividad, la cámara y la comunicación con el backend.

4. **Diseño e implementación del servidor web**
   Se desarrolló una API REST con **Node.js** y una interfaz web sencilla para la administración de credenciales y logs del sistema.

5. **Integración progresiva del sistema completo**
   Una vez probados los módulos por separado, se integró la comunicación **Arduino <-> ESP32** vía serial, y posteriormente **ESP32 <-> Backend** vía HTTP. También se conectó el bot de Telegram para el envío de alertas.

6. **Montaje físico y construcción de la maqueta**
   En paralelo a la integración de todos los elementos, se diseñó y montó una **maqueta física** representativa de una puerta con sus sensores y actuadores integrados.

7. **Documentación y preparación de la presentación**
   Finalmente, se elaboró la memoria técnica, el repositorio con el código, y se preparó la presentación del proyecto.

4. Problemas/Desafíos y sus soluciones

Durante el desarrollo del proyecto, nos encontramos con diversos problemas que requirieron soluciones técnicas y de diseño. A continuación, detallamos los principales desafíos a los que nos enfrentamos y cómo los resolvimos:

1. **Calidad de los materiales**
   Algunos componentes, como ciertos cables o módulos, presentaban una calidad deficiente que dificultaba las conexiones o provocaba fallos intermitentes. Para solventarlo, optamos por soldar aquellos módulos que nos causaban problemas como el lector NFC. Para soldar los módulos hay que hacerlo con la mayor precisión posible ya que puede hacer que el módulo deje de funcionar

2. **Imprecisión del servomotor**
   El servomotor encargado del cierre y apertura de la puerta no realizaba un giro exacto de 90 grados, lo cual afectaba el correcto funcionamiento del sistema. Para resolver este problema tuvimos que realizar ajustes en el código a base de ensayo y error hasta encontrar el porcentaje justo que le teníamos que dar para que hiciera el giro correctamente. Además, si la maqueta se encuentra inclinada afectará a este giro, haciendo que posiblemente la puerta no realice el giro de manera adecuada.

3. **Mal montaje de los componentes**
   Inicialmente, algunas conexiones entre módulos no eran del todo claras, lo que ocasionaba errores de comunicación o fallos eléctricos. Para solucionarlo optamos por utilizar colores específicos para determinadas conexiones, por ejemplo el color **rojo** para todo lo relacionado con VCC o el **negro** para GND.

4. **Montaje de la puerta de la maqueta**
   Diseñar y montar una puerta funcional a escala resultó más complejo de lo previsto, especialmente para lograr que se integrara bien con el servomotor y

que realice el giro de manera correcta. Como solución a este problema tuvimos que tener en cuenta que el elemento que gira del servomotor tiene que estar en el eje de giro de la puerta.

5. **Problemas con la comunicación Bluetooth**
   Antes de tener el área de seguridad con sensor de movimiento (PIR) nuestra idea era que este área fuese bluetooth de esta forma si una persona no autorizada entra en el área sonaría la alarma directamente. Cuando nos pusimos a implementar esta funcionalidad empezaron a surgir varios problemas. El primero es que existe dos tipos de Bluetooth:

   - **Bluetooth clásico**: El más antiguo y usado para conexiones más estables que permite una transferencia de datos más alta.
   - **Bluetooth LE** (low energy): Es más actual y menos costoso, la mayoría de nuestros móviles lo tienen y funciona escaneando cada cierto tiempo el área bluetooth buscando dispositivos que estén mandando señales.

   La ESP32 permite tanto Bluetooth clásico como LE y los que buscábamos era que la propia ESP32 escaneara el área Bluetooth buscando MACs autorizadas, pero los móviles actuales por seguridad cambian la MAC cada cierto tiempo por lo tanto era inviable que fuese persistente. La siguiente opción era tratar a la ESP32 como si fuese un teclado o unos cascos, de esta forma si que se conectaba pero limita las funciones de la ESP32, tras estos dos problemas decidimos probar con el clásico, conseguimos conectarlo aunque solo nos permitía tener un solo dispositivo conectado, pero ya teníamos un posible solución a explorar.

6. **RAM ESP32**
   Una vez que ya teníamos una solución para el Bluetooth, apareció uno nuevo, y es que la ESP32 no tenía suficiente **RAM** para realizar todas las funcionalidades ya que las librerías eran bastante pesadas. Por temas de tiempo y imposibilidad para comprar algún dispositivo para tener más RAM decidimos cambiar este área bluetooth por el área con sensor de movimiento. De todos modos se solucionaría aumentando la RAM de la ESP32.

7. **Estabilidad de la comunicación entre dispositivos**
   El último problema que tuvimos fue la comunicación entre el **Servidor Web** y **la ESP32** y la **ESP32** con la **Arduino**, es decir la ESP32 haría de puente entre ambos. El primer problema que tuvimos es que el traspaso de información del Servidor a la ESP32 era el correcto pero de la ESP32 al Arduino no, esto es

causado porque la ESP32 trabaja a 3.3V y la Arduino a 5V por lo que tenemos que aumentarlo a 5V.

Además apareció otro problema, la ESP32 no se conectaba con el servidor, la solución consistía en tener todo en la misma red wifi, además, es muy importante que la red wifi sea una de **confianza**, esta opción se cambia en el configurador de la red. Lo más cómodo y recomendable es crear una red wifi propia con los datos del móvil y conectarlo todo ahí configurando la red desde el ordenador donde esté conectado el Servidor Web.

5. Resumen del proyecto

Ring Door es un sistema de control de acceso multifactor y seguridad, diseñado para proporcionar una solución robusta y monitorizada para la gestión de entradas físicas. El sistema se basa en la interacción de un microcontrolador ESP32-CAM, que actúa como unidad central, y un microcontrolador Arduino, que gestiona los periféricos de bajo nivel. Complementado por un servidor backend y una interfaz web, RingDoor permite la autenticación mediante tarjetas NFC y reconocimiento de huellas dactilares, ofrece monitorización en tiempo real con captura de imágenes y envía notificaciones a través de Telegram ante eventos relevantes.

Este documento detalla la arquitectura, diseño funcional, especificaciones técnicas, flujos de operación y consideraciones de implementación del sistema Ring Door.

6. Arquitectura del sistema

La arquitectura de Ring Door se articula en torno a tres módulos principales que interactúan para proporcionar las funcionalidades deseadas:

- **Controlador ESP32-CAM**: Es el cerebro del sistema, gestionando la conectividad WiFi, la comunicación con el servidor backend (API), la interacción con el módulo Arduino, la captura de imágenes a través de su cámara integrada y el envío de notificaciones a Telegram. El código principal se encuentra en el archivo main.ino (para ESP32).

- **Controlador Arduino (UNO/Nano o similar)**: Se encarga de la gestión directa de los periféricos de bajo nivel. Esto incluye la lectura de tarjetas NFC (MFRC522), el registro y verificación de huellas dactilares (sensor R503), el control de la cerradura física (mediante un servo motor), la lectura del sensor de movimiento (PIR), y la interacción con el usuario a través de una pantalla OLED y un buzzer. El código principal se encuentra en el archivo main.ino (para Arduino).
- **Servidor Backend**: Una aplicación Node.js (definida en backend/server.js) que expone una API RESTful. Se encarga de la persistencia de datos (usuarios, UIDs de tarjetas NFC, registros de huellas, logs de acceso y seguridad) y sirve la interfaz web para la administración del sistema.

6.1. Diagrama de Bloques de la Arquitectura

El flujo de datos principal se puede conceptualizar de la siguiente manera:

1. El **Usuario** interactúa con el sistema presentando una tarjeta **NFC** o su **Huella Dactilar** al **Controlador Arduino**.
2. El **Arduino** procesa la entrada y, según el evento (ej. acceso concedido, intento fallido, detección de movimiento), envía un **Evento/Comando** específico al **Controlador ESP32-CAM** vía comunicación serial.
3. El **ESP32-CAM** recibe el comando, lo procesa y puede realizar varias acciones:
    - Enviar **Datos/Alertas** (logs, solicitud de foto, etc) a la **API del Servidor Backend**.
    - Enviar notificaciones (con o sin imagen) a **Telegram**.
    - Enviar **Comandos/Actualizaciones** (ej. lista de UIDs NFC actualizados) de vuelta al **Arduino**.
4. El **Servidor Backend** interactúa con el **Frontend** (interfaz web) para la administración y visualización de datos por parte del administrador.

6.2. Componentes de Hardware

1. **Unidad de Comunicación y Cámara (ESP32-CAM AI-Thinker o similar)**:
    - Microcontrolador con WiFi, Bluetooth (no usado activamente en la versión actual del código ESP32) y cámara integrada (OV2640).
    - Referencia de código: esp32/main.ino
2. **Unidad de Control de Periféricos (Arduino UNO R3)**:
    - Microcontrolador para manejo de sensores y actuadores.

    - Referencia de código: arduino/main.ino
3. **Conversor de Nivel Lógico Bidireccional (3.3V <-> 5V)**:
    - Esencial para la comunicación serial segura entre el ESP32-CAM (3.3V) y el Arduino (5V).(En nuestro caso no nos dio tiempo a comprarlo antes de la presentación)
4. **Periféricos Conectados al Arduino**:
    - **Lector NFC (MFRC522)**: Para lectura de tarjetas/tags NFC (ISO 14443A), conectado vía SPI.
    - **Sensor de Huellas Dactilares (AS608)**: Para autenticación biométrica. Conectado vía SoftwareSerial.
    - **Servo Motor (SG90)**: Para accionar el mecanismo de la cerradura.
    - **Sensor de Movimiento Infrarrojo Pasivo (PIR HC-SR501)**: Para detectar una presencia.
    - **Pantalla OLED (SSD1306, 128x32)**: Para mostrar el estado y mensajes. Conectada vía I2C.
    - **Buzzer Activo/Pasivo**: Para la simulación de alarma.
5. **Periféricos del ESP32-CAM**:
    - **Módulo de Cámara**: Integrado, configurado para JPEG.
    - **Antena WiFi**: Integrada.

6.3. Componentes de Software

1. **Firmware Arduino**:
    - Desarrollado en C/C++ (framework Arduino).
    - Gestiona sensores, actuadores, OLED y comunicación serial con ESP32.
2. **Firmware ESP32-CAM**:
    - Desarrollado en C/C++ (framework ESP-IDF para Arduino).
    - Maneja WiFi, cámara, comandos del Arduino, interacción con API backend y Telegram.
3. **Servidor Backend**:
    - Node.js con Express.js (backend/server.js).
    - API RESTful para gestión de datos y servicio del frontend.
    - Persistencia de datos: Archivos JSON en backend/data/.
4. **Aplicación Frontend**:
    - HTML, CSS, JavaScript (frontend/).
    - Interfaz para login, gestión de credenciales, visualización de logs y fotos.
5. **Scripts Utilitarios**:
    - arduino/clear-fingerprints.ino: Para borrar huellas del sensor R503.

- backend/setup.sh, backend/start.sh (y equivalentes .bat): Para dependencias e inicio del backend.

6.4. Protocolos de Comunicación

1. **Arduino <-> ESP32-CAM**: Serial asíncrona bidireccional.
   - Arduino: se comunica mediante SoftwareSerial espSerial (Pines 8-RX, 5-TX).
   - ESP32-CAM: se comunica mediante HardwareSerial ArduinoSerial(2) (Pines 14-RX, 15-TX).
   - Requiere conversor de nivel lógico para pasar de los 3.3V de la ESP32 a los 5V del Arduino.
   - Baud rate: 9600 bps (configurable, ARDUINO_BAUD_RATE).
   - Formato: Cadenas ASCII terminadas en \n.
2. **ESP32-CAM <-> Servidor Backend**: HTTP (principalmente) sobre TCP/IP (WiFi).
   - ESP32-CAM actua como cliente HTTP (HTTPClient).
   - URL base: API_BASE_URL.
3. **ESP32-CAM <-> API Telegram**: HTTPS sobre TCP/IP (WiFi).
   - ESP32-CAM usa la librerías WiFiClientSecure telegramClientSecure y UniversalTelegramBot, para comunicarse con Telegram.
   - Token: TELEGRAM_BOT_TOKEN, Chat ID: TELEGRAM_CHAT_ID.
4. **Frontend <-> Servidor Backend**: HTTP sobre TCP/IP.

7. Requisitos del sistema

7.1. Requisitos Funcionales

- **RF001:** Registro y eliminación de tarjetas NFC.
- **RF002:** Registro (asociado a tarjeta NFC) y eliminación de huellas dactilares.
- **RF003:** Concesión de acceso (activación de servo) mediante huella dactilar válida.
- **RF004:** Visualización de mensajes de estado en pantalla OLED.
- **RF005:** Activación de buzzer ante múltiples intentos fallidos de acceso.
- **RF006:** Detección de movimiento (PIR).
- **RF007:** Captura de foto y envío a Telegram tras N fallos de huella.

- **RF008:** Notificación a Telegram tras detección de movimiento.
- **RF009:** Almacenamiento de todos los eventos de acceso y seguridad en el backend.
- **RF010:** Sincronización periódica de UIDs NFC autorizados desde el backend al ESP32 y luego al Arduino.
- **RF011:** Check-in periódico del ESP32 al backend.
- **RF012:** Interfaz web para administración de usuarios, logs y fotos.
- **RF013:** Autenticación para acceso a la interfaz web.
- **RF014:** Autorización manual de tarjetas NFC no reconocidas (vía serial Arduino, iniciada remotamente).

7.2. Requisitos No Funcionales

- **RNF001 (Rendimiento):** Verificación de huella < 2s. Lectura NFC < 1s. Notificación Telegram < 15s (dependiente de red).
- **RNF002 (Fiabilidad):** Operación continua. Comunicación serial robusta.
- **RNF003 (Usabilidad):** Interfaz web intuitiva. Mensajes en la pantalla OLED claros.
- **RNF004 (Seguridad):** Comunicación Telegram vía HTTPS. Protección de tokens/credenciales.
- **RNF005 (Mantenibilidad):** Código comentado y modularizado.
- **RNF006 (Escalabilidad):** Backend con JSON.

8. Funcionalidades principales detalladas

8.1. Autenticación Multifactor

- **Tarjeta NFC**:
  - Arduino (arduino/main.ino) detecta una tarjeta NFC. Verifica su UID con authorizedCards.
  - Si no está autorizada: se autoriza con handleNFCAuthorization (serial Arduino) o enviando el UID a la ESP32.
  - ESP32 (esp32/main.ino -> handleArduinoCommands): con el comando NFC_UID_AUTH:UID_HEX añade el UID a la whitelistNfcUids y lo envía al backend (sendNfcUidToApi a /api/esp32/cards).
  - Sincronización: ESP32 (fetchNfcUids desde /api/cards/uids) le manda los UIDs actualizados a la Arduino (sendNfcUpdateToArduino con el comando NFC_UPDATE:UID1;UID2...).

- **Huella Dactilar**:
    - Registro (arduino/main.ino -> registrarHuellas): Iniciado tras presentar NFC autorizada, guiado por la pantalla OLED. Arduino envía el ID a la ESP32 a través de REGISTER:ID HUELLA;ID TARJETA.
    - ESP32 (handleArduinoCommands) recibe el ID (REGISTER) y llama a /api/register-fingerprint en backend.
    - Verificación: Arduino compara la huella, sii es una huella correcta envía ACCESS_GUARANTEED:ID_HUELLA a la ESP32.

8.2. Monitorización de Seguridad

- **Detección de Movimiento**: la Arduino (checkPIR) envía MOTION_DETECTED a la ESP32. La ESP32 maneja el flag (handleArduinoCommands) envía el mensaje sendMessageTelegram y sendMotionDetectedLog (a /api/security-logs).
- **Captura de Fotografías**: Arduino (fallo huella) -> PHOTO_REQUEST_FP_FAIL (ESP32 espera PHOTO) a ESP32. ESP32 (takePhotoAndSendAlerts) -> captura, envía a Telegram y a backend (/api/security-logs/upload-photo).
- **Registro de Eventos (Logs)**: ESP32 envía logs al backend:
    - Acceso por huella: sendFingerprintLogToBackend.
    - Movimiento: sendMotionDetectedLog.
    - Fallo con foto: takePhotoAndSendAlerts.
- **Notificaciones en Tiempo Real**: Alertas a Telegram (TELEGRAM_CHAT_ID, TELEGRAM_BOT_TOKEN).

8.3. Gestión Remota (Backend y Frontend)

- Backend (backend/server.js) expone API.
- Frontend (frontend/) permite: visualización de logs, gestión de credenciales (inferido).

8.4. Control de Acceso Físico

- Autenticación OK -> Arduino (abrirCerradura) activa el servo.
- Feedback: a través de la pantalla OLED (displayState) y el Buzzer (activateBuzzer).

9. Flujo de datos y comunicación

9.1. Arduino <=> ESP32-CAM

- **Medio**: Comunicación serial (SoftwareSerial en Arduino, HardwareSerial en ESP32).
- **Comandos Arduino -> ESP32** (procesados en handleArduinoCommands de la ESP32):
  - MOTION_DETECTED (Arduino envía MOTION)
  - PHOTO_REQUEST_FP_FAIL (Arduino envía una flag de que se ha se ha fallado tres veces al intententar entrar PHOTO_REQUEST_FP_FAIL, ESP32 espera la PHOTO)
  - ACCESS_GUARANTEED:ID_HUELLA
  - NFC_UID_AUTH:UID_HEX_CON_DOS_PUNTOS
  - REGISTER:ID_HUELLA;UID_TARJETA_HEX_CON_DOS_PUNTOS
- **Comandos ESP32 -> Arduino** (procesados en handleSerialCommands del Arduino, que debería ser handleEsp32Commands):
  - BUZZER_ON / BUZZER_OFF
  - NFC_UPDATE:UID1_HEX_CON_DOS_PUNTOS;UID2_HEX_CON_DOS_PUNTOS.

9.2. ESP32-CAM <=> Servidor Backend (API)

- **Medio**: HTTP/HTTPS. ESP32 usa HTTPClient.
- **Endpoints API clave**:
  - POST formato: {API_BASE_URL}/{API_CHECKIN_ENDPOINT} (periodicCheckin)
  - GET formato: {API_BASE_URL}/{API_NFC_UIDS_ENDPOINT} (fetchNfcUids)
  - GET formato: {API_BASE_URL}/{API_PING_ENDPOINT}
  - POST formato: {API_BASE_URL}/api/esp32/cards (sendNfcUidToApi)
  - POST formato: {API_BASE_URL}/api/register-fingerprint
  - POST formato: {API_BASE_URL}/api/security-logs
  - POST formato: {API_BASE_URL}/api/security-logs/upload-photo

- **Formato de Datos**: JSON.

9.3. ESP32-CAM <=> Telegram

- **Medio**: HTTPS (WiFiClientSecure, UniversalTelegramBot).
- Envía mensajes y fotos.

9.4. Frontend <=> Servidor Backend

- La interfaz web (frontend/script.js) usa AJAX/Fetch API para interactuar con el backend. fetchWithAuth permite autenticación de sesión con token.

10. Detalles de implementación del software

10.1. ESP32-CAM (esp32/main.ino)

- **Setup**: Inicializa el hardware serial, PSRAM, WiFi, la cámara ajustando sus parametros (initCamera ajusta la resolución FRAMESIZE_UXGA/FRAMESIZE_XGA y la calidad), ArduinoSerial, cliente HTTPS, el Telegram y timers para hacer consultas al servidor web (periodicApiUpdate, periodicCheckin).
- **Loop**: Maneja la reconexión WiFi, procesa flags que le llegan del Arduino handleArduinoCommands.
- **Tareas Periódicas**: periodicApiUpdate (obtiene UIDs NFC y los envía a Arduino) y periodicCheckin (ping al backend para no perder la conexió por inactividad).

10.2. Arduino (arduino/main.ino)

- **Setup**: Inicializa los periféricos (OLED, huella, NFC, servo, PIR, buzzer)y el software serial de la espSerial.
- **Loop**: Verifica NFCs, huellas, PIR y comandos de espSerial.
- **Lógica Autenticación**: comprueba si una NFC está autorizada (isAuthorized) registra nuevas huellas (registrarHuellas).

10.3. Servidor Backend (backend/server.js)

- API RESTful Node.js/Express.js.
- Endpoints para realizar el check-in, obtener y registrar nuevas UIDs, registro de huellas, logs y subida de fotos.
- Persistencia en archivos JSON.

10.4. Frontend (frontend/)

- HTML, CSS, JS (script.js con lógica de API y fetchWithAuth).

11. Configuración y puesta en marcha

1. **Clonar Repositorio**: git clone 'https://github.com/Eclipse3k/RingDoor.git'
2. **Backend**: cd backend, ejecutar setup.sh (o .bat), luego start.sh (o .bat).
3. **ESP32 (esp32/main.ino)**: Modificar WIFI_SSID, WIFI_PASSWORD, API_BASE_URL, TELEGRAM_BOT_TOKEN, TELEGRAM_CHAT_ID. Cargar firmware.
4. **Arduino (arduino/main.ino)**: Cargar firmware.
5. **(Opcional)** arduino/clear-fingerprints.ino para limpiar sensor.
6. **Acceder a Interfaz Web**: http://<IP_BACKEND>:3000.

12. Consideraciones de seguridad

- clientSecure.setInsecure() en ESP32 para HTTPS (si API_BASE_URL fuera HTTPS) debe evitarse en producción, manejando certificados SSL/TLS.
- La comunicación con Telegram usa telegramClientSecure.setCACert(TELEGRAM_CERTIFICATE_ROOT)
- Credenciales WiFi/Tokens: El almacenamiento seguro es crucial (NVS en la ESP32 y variables de entorno en backend).
- API Backend: Implementar autenticación/autorización robusta.

13. Conclusión y posibles mejoras futuras

Ring Door ha sido un proyecto que aunque no ha sido el más original entre todos los que se han presentado nos ha permitido ver la complejidad que tienen los sistemas de vigilancia y seguridad de la actualidad. Consideramos que, para el tiempo que disponíamos, hemos conseguido construir un sistema bastante comparable a los actuales y nos sentimos muy gratificados con el resultado obtenido, además esta experiencia nos ha permitido adentrarnos en el mundo de la programación de dispositivos físicos, un ámbito que a menudo pasa desapercibido frente al desarrollo de software convencional, pero que resulta igual de desafiante e interesante. Por último nombramos algunas mejoras que se podrían realizar a futuro para mejorar nuestro sistema.

Posibles Mejoras Futuras:

- **Base de Datos Robusta:** Migrar de JSON a SQL/NoSQL para el backend.
- **Comunicación Segura (HTTPS):** End-to-end para ESP32-Backend.
- **Autenticación Mejorada:** OAuth2/JWT para frontend y API.
- **Configuración Dinámica del ESP32:**Tratamiento de credenciales mediante Bluetooth.
- **Integración Domótica:** MQTT u otros protocolos.
- **Anti-Tampering:** Detección de manipulación física.