UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

Sistema de identificación del trabajador social en la vivienda

Autor:

D. Adolfo José Dopico González

Tutor/es:

Dr. D. Alfonso Bahillo Martínez

Valladolid, Julio de 2023

TÍTULO:	**Sistema de identificación del trabajador social en la vivienda**
AUTOR:	**D. Adolfo José Dopico González**
TUTOR:	**Dr. D. Alfonso Bahillo Martínez**
DEPARTAMENTO:	**Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

TRIBUNAL

PRESIDENTE:	**Dr. D. Ramón J. Durán Barroso**
VOCAL:	**Dr. D. Ignacio de Miguel Jiménez**
SECRETARIO:	**Dr. D. Juan Carlos Aguado Manzano**
SUPLENTE 1:	
SUPLENTE 2:	

FECHA:	**01/07/2025**
CALIFICACIÓN:	

Agradecimientos

En primer lugar, quiero agradecer a mi tutor, Alfonso, por su plena disponibilidad en todo momento, su trato cercano y sincero, y por ayudarme siempre que lo he necesitado a lo largo del desarrollo de este trabajo. Gracias por tu confianza y por tus valiosos consejos, que han sido clave para llevar a buen puerto este proyecto.

También quiero expresar mi agradecimiento a todos los trabajadores que han participado en las pruebas piloto, utilizando la aplicación en su día a día y proporcionando una realimentación continua y constructiva. Sus observaciones y sugerencias han sido fundamentales para pulir detalles, resolver incidencias y lograr que el sistema sea más robusto, funcional y adaptado a sus necesidades reales.

Con este trabajo concluyo una etapa muy importante en mi formación como Ingeniero de Telecomunicación en el marco del Máster Universitario en Ingeniería de Telecomunicación. Aprovecho la ocasión para dar las gracias a todo el profesorado que ha hecho posible este recorrido, transmitiéndonos sus conocimientos y experiencia de la mejor manera posible, y a mis compañeros y amigos del máster, que han estado a mi lado en los buenos y en los malos momentos, consiguiendo que este camino fuera mucho más ameno y llevadero.

Gracias también a mis amigos de toda la vida por ofrecerme esa distracción tan necesaria cuando las cosas se complicaban, y a mi familia —en especial a mi madre y a mis abuelos— por su apoyo incondicional, por su cariño y por estar siempre ahí incluso cuando yo no era consciente de que lo necesitaba.

Por último, gracias a mi pareja, por ser un pilar fundamental en mi vida y por ayudarme a crecer tanto como persona como en mi faceta de estudiante. Sin su apoyo constante, su paciencia y su motivación, este proyecto no habría sido el mismo.

Resumen

Este Trabajo Fin de Máster aborda el diseño y desarrollo de un sistema de verificación automática de presencia laboral destinado a mejorar la gestión de los servicios de atención domiciliaria prestados por la Fundación Intras. La organización, orientada al apoyo de personas en situación de dependencia, planteó la necesidad de implantar un mecanismo que permitiera comprobar de forma objetiva y no intrusiva que sus trabajadores se encuentran efectivamente en los domicilios de los usuarios durante los horarios asignados.

Para dar respuesta a este reto, se ha desarrollado una solución basada en el uso de tecnologías inalámbricas de bajo consumo, como Bluetooth Low Energy, integrada en una arquitectura distribuida que aprovecha la infraestructura tecnológica ya existente en la fundación: dispositivos móviles Android utilizados por los trabajadores y *Gateways* con sistema operativo Debian 11, instalados en cada vivienda. Ambos componentes se comunican indirectamente a través de un servidor central implementado con MQTT, que coordina los horarios de emisión y escaneo.

El sistema consta de dos aplicaciones principales. Por un lado, una aplicación móvil desarrollada en Kotlin que se conecta al servidor, recibe la planificación horaria y emite señales BLE únicamente durante el horario laboral asignado. Por otro lado, un *script* en Python que se ejecuta en el *Gateway* y realiza escaneos periódicos para detectar las señales de los dispositivos autorizados. La detección se filtra mediante un identificador de empresa y el ID del dispositivo móvil, registrando así la presencia del trabajador.

Adicionalmente, se ha desarrollado una tercera versión que sustituye la aplicación móvil por una baliza Bluetooth física que emite señales en formato *Eddystone*. Esta baliza incluye en su campo *Service Data* los identificadores de empresa y dispositivo, permitiendo que el *Gateway* la detecte y filtre de manera autónoma, eliminando la dependencia de un *smartphone* y reduciendo la intervención del trabajador.

Durante el proyecto se han abordado aspectos técnicos como la configuración del hardware, la compatibilidad de adaptadores Bluetooth, la automatización del escaneo y el almacenamiento local de registros. Asimismo, se ha tenido en cuenta el cumplimiento de criterios éticos y de privacidad, garantizando que el sistema solo opere dentro del horario laboral y que no almacene información personal sensible.

Como resultado, se ha obtenido un sistema funcional, autónomo y eficiente que permite a la Fundación Intras mejorar la trazabilidad de los servicios, optimizar la planificación del personal y reforzar la transparencia ante familias y organismos supervisores. Además, el trabajo sienta las bases para futuras mejoras, como la gestión centralizada de los registros, el cifrado de las comunicaciones o la expansión a nuevos dispositivos y domicilios.

Palabras clave:

Bluetooth, Gateway, MQTT, script, registro, adaptador.

Abstract

This Master's Thesis presents the design and development of an automatic presence verification system aimed at improving the management of home care services provided by Fundación Intras. The organization, dedicated to supporting individuals in situations of dependency, identified the need for a mechanism to objectively and non-intrusively confirm that its workers are physically present at users' homes during their assigned working hours.

To address this challenge, a solution was implemented based on low-energy wireless technologies, such as Bluetooth Low Energy, integrated into a distributed architecture that leverages the organization's existing infrastructure: Android mobile devices used by employees and Debian 11–based Gateways installed in each home. Both components communicate indirectly through a central MQTT server, which synchronizes the emission and scanning schedules.

The system comprises two main applications. On one side, a mobile app developed in Kotlin connects to the server, receives the scheduled working hours, and emits BLE signals only during the specified time window. On the other side, a Python script running on the Gateway periodically scans for BLE signals and filters that match the authorized devices based on a company ID and the unique Android ID.

Additionally, a third version was developed in which the mobile application is replaced by a standalone Bluetooth beacon. This beacon broadcasts signals in the Eddystone format and encodes the company and device identifiers into its Service Data field, allowing the Gateway to detect and filter it without relying on a smartphone and further reducing the worker's required interaction.

Throughout the project, technical aspects such as hardware setup, Bluetooth adapter compatibility, automated scanning, and local data logging were carefully addressed. Ethical and privacy considerations were also considered, ensuring that the system only operates within working hours and does not store personal data.

The outcome is a fully functional, autonomous, and efficient system that enables Fundación Intras to enhance service traceability, optimize workforce scheduling, and increase transparency with families and supervisory bodies. Moreover, the project lays the groundwork for future improvements, such as centralized record management, encrypted communications, or scalable deployment across additional homes and devices.

Keywords:

Bluetooth, Gateway, MQTT, script, registro, dongle.

ÍNDICE GENERAL

Índice de siglas y acrónimos

BLE	*Bluetooth Low Energy*
GNSS	*Sistema Global de Navegación por Satélite*
IDE	*Integrated Development Environment*
IoT	*Internet of Things*
MQTT	*Message Queuing Telemetry Transport*
NFC	*Near Field Communication*
QR	*Quick Response*
RGPD	*Reglamento General de Protección de Datos*
TIC	*Tecnologías de la Información y la Comunicación*
TFM	*Trabajo de Fin de Máster*
UVa	*Universidad de Valladolid*

Capítulo 1: INTRODUCCIÓN

El presente Trabajo Fin de Máster (TFM) surge a raíz de una necesidad concreta planteada por la Fundación Intras (https://www.intras.es/), entidad dedicada a la atención de personas en situación de dependencia. A partir de esta colaboración, se ha definido un proyecto que busca dar solución a un problema real: la verificación fiable, automática y no intrusiva de la presencia de los trabajadores en los domicilios donde prestan servicios. Este apartado introductorio contextualiza dicha necesidad desde diferentes perspectivas, comenzando por el papel que desempeñan las tecnologías móviles en la actualidad, seguido de una presentación institucional de la Fundación Intras, y finalizando con el planteamiento específico del proyecto, su infraestructura base y la motivación que lo justifica.

1.1. Contexto y motivación

En las últimas décadas, el avance imparable de la tecnología ha transformado de manera radical la forma en que las personas se comunican, trabajan y acceden a servicios. Un aspecto especialmente relevante de esta transformación ha sido la evolución de los dispositivos móviles, que han pasado de ser simples herramientas de comunicación a convertirse en auténticas plataformas multifuncionales. Gracias a la integración de capacidades como la geolocalización, la conectividad inalámbrica, los sensores y las aplicaciones de gestión, los dispositivos móviles se han consolidado como aliados indispensables en el ámbito profesional, donde permiten optimizar procesos, mejorar la coordinación de equipos y garantizar la calidad de los servicios prestados. En el sector de la asistencia domiciliaria, los dispositivos móviles han adquirido un papel clave como herramienta de coordinación y seguimiento de los profesionales, así como para verificar su presencia en los domicilios de los usuarios.

La Fundación Intras es una entidad sin ánimo de lucro dedicada a promover la inclusión social y la mejora de la calidad de vida de personas en situación de vulnerabilidad, especialmente aquellas con problemas de salud mental, discapacidad intelectual y personas mayores. Desde su creación en 1994, la Fundación ha centrado sus esfuerzos en diseñar e implementar programas de atención individualizada, basados en un enfoque integral que considera no solo las necesidades médicas, sino también las dimensiones sociales, psicológicas y emocionales de cada persona. Su misión es ofrecer servicios que permitan a las personas mantener la mayor autonomía posible y disfrutar de una vida digna y participativa, promoviendo al mismo tiempo la sensibilización social y el respeto por la diversidad. [1]

Figura 1: Logotipo Fundación Intras [1]

A través de sus diferentes proyectos y centros de trabajo, la Fundación Intras desarrolla actividades de atención domiciliaria, empleo protegido, formación, ocio y apoyo psicosocial, siempre con el objetivo de garantizar la inclusión y la mejora continua de la calidad de vida de sus beneficiarios. Su equipo multidisciplinar, compuesto por profesionales especializados en trabajo social, psicología, enfermería, terapia ocupacional y otras disciplinas, trabaja de manera coordinada para ofrecer servicios de atención y acompañamiento adaptados a las necesidades de cada persona.

En un contexto marcado por el envejecimiento progresivo de la población y el aumento de las necesidades de atención personalizada, surge la necesidad de adoptar soluciones tecnológicas que permitan garantizar la calidad y el correcto desarrollo de los servicios asistenciales. Uno de los principales retos en este sentido es el control de la presencia de los trabajadores en los domicilios de las personas atendidas, dado que la dispersión geográfica de estos domicilios y la necesidad de verificar el cumplimiento de los horarios establecidos suponen desafíos organizativos y de gestión.

La Fundación Intras, consciente de esta necesidad, se ha propuesto implantar un sistema de registro que permita verificar, de manera objetiva y en tiempo real, la presencia de sus empleados en los domicilios de los beneficiarios. Este sistema pretende convertirse en una herramienta de gestión eficaz que facilite la supervisión de los servicios prestados, optimice la planificación de los recursos y garantice la transparencia y la calidad asistencial.

Un aspecto fundamental en el diseño e implementación del sistema de registro es garantizar el respeto a la privacidad y los derechos de los empleados. Según el Reglamento General de Protección de Datos (RGPD) y la Ley Orgánica 3/2018 de Protección de Datos Personales y garantía de los derechos digitales, el tratamiento de datos personales, como la localización mediante Bluetooth, requiere cumplir con los principios de licitud, transparencia y minimización de datos. [2] [3]

En este sentido, es indispensable que los trabajadores sean informados de forma clara sobre la finalidad del tratamiento de sus datos y que se limite exclusivamente a verificar su presencia en los domicilios para garantizar la calidad del servicio. Asimismo, deben establecerse medidas de seguridad adecuadas para proteger la confidencialidad y la integridad de la información.

Actualmente, las empresas del sector de la asistencia domiciliaria utilizan diversos métodos de registro para el control de presencia de sus empleados. Entre los más habituales se encuentran las aplicaciones móviles basadas en geolocalización, los sistemas de registro manual mediante códigos QR (Quick Response) o contraseñas y, en menor medida, las soluciones basadas en NFC o Bluetooth. Cada uno de estos métodos presenta ventajas e inconvenientes: mientras que las soluciones basadas en GNSS (Sistema Global de Navegación por Satélite) pueden verse afectadas por problemas de precisión en interiores o entornos urbanos, los sistemas manuales requieren una intervención activa del trabajador, lo que puede conllevar errores o fraudes involuntarios. Por su parte, las soluciones basadas en Bluetooth ofrecen un equilibrio entre automatización, fiabilidad y facilidad de integración con la infraestructura ya existente, lo que las convierte en una opción especialmente interesante para la Fundación Intras.

1.2. Introducción al proyecto

La Fundación Intras cuenta con una infraestructura tecnológica ya instalada en los domicilios de sus clientes: cada hogar dispone de un *Gateway* con el sistema operativo Debian 11, que permite la implementación de soluciones de software adaptadas y la conectividad con dispositivos móviles. Estos dispositivos, ubicados estratégicamente en las viviendas, actúan como puntos de control y comunicación entre el trabajador y la organización. Asimismo, los profesionales de la Fundación utilizan teléfonos corporativos con sistema operativo Android, lo que ofrece una base idónea para la integración de aplicaciones móviles de verificación de presencia mediante tecnologías inalámbricas como Bluetooth. Esta combinación de infraestructura en el hogar y dispositivos móviles corporativos proporciona un marco técnico robusto y escalable para la implantación de soluciones innovadoras que garanticen la calidad y la transparencia en la prestación del servicio.

Para dar respuesta a la necesidad de garantizar la correcta prestación de sus servicios, el presente proyecto se plantea como el diseño y desarrollo de un sistema de registro que, a través de anuncios y escaneos BLE (Bluetooth Low Energy), permita detectar de manera automática la presencia de los trabajadores en el rango de cobertura del *Gateway* de cada domicilio. Este sistema permitirá registrar de forma digital y en tiempo real tanto la entrada como la salida de los empleados, generando evidencia objetiva de la prestación del servicio. Con ello, se persigue optimizar la planificación de las tareas, reforzar la supervisión y la transparencia en la gestión de la atención domiciliaria, respetando en todo momento la privacidad y la autonomía de los profesionales y de las personas atendidas.

La implementación de este sistema se complementa con un servidor MQTT como elemento central de sincronización y coordinación remota. Este servidor permitirá a la Fundación Intras gestionar de manera centralizada los horarios de trabajo de cada empleado y enviar las instrucciones necesarias tanto a la aplicación móvil como al *Gateway*, asegurando que la emisión y el escaneo de anuncios BLE

se realicen únicamente durante los periodos asignados. Esta arquitectura distribuida facilita la administración y supervisión de los servicios desde la propia Fundación, minimizando las tareas de configuración local y permitiendo adaptaciones ágiles y escalables a medida que evolucionen las necesidades de la organización o de los usuarios.

En definitiva, esta solución tecnológica aprovecha la infraestructura existente y las posibilidades de la tecnología Bluetooth y MQTT para dotar a la Fundación Intras de una herramienta moderna y eficiente, alineada con su compromiso social y con las exigencias de calidad y profesionalización del sector de la atención domiciliaria.

1.3. Fases de desarrollo del proyecto

El desarrollo del proyecto se estructuró en tres fases principales, que permitieron abordar de forma progresiva tanto la implementación local en los dispositivos como la integración de la comunicación entre ellos mediante un servidor central.

1.3.1. Primera fase: Desarrollo de la aplicación móvil y el script de escaneo

En esta etapa inicial, se diseñó y desarrolló una aplicación móvil específica destinada a los empleados de la Fundación Intras. Su principal objetivo es permitir a los trabajadores planificar y registrar sus jornadas laborales de forma sencilla y práctica. La aplicación permite a cada empleado introducir los días y los horarios en los que prestará servicio en el domicilio de un cliente, generando así un calendario de trabajo personalizado.

Durante los intervalos de tiempo definidos por el usuario, la aplicación se encarga de emitir anuncios BLE, actuando como una "baliza" que confirma la presencia del trabajador en el domicilio. Esta funcionalidad convierte al dispositivo móvil en una pieza clave del sistema de verificación de presencia, aprovechando su portabilidad y las capacidades de conectividad Bluetooth integradas en la mayoría de los *smartphones* actuales.

En paralelo, se desarrolló un *script* destinado a ejecutarse en el *Gateway* instalado en el domicilio del usuario. Este *Gateway*, basado en Debian 11, constituye la infraestructura tecnológica de cada vivienda donde se prestan los servicios. El *script* está diseñado para escanear el entorno en busca de las señales BLE emitidas por las aplicaciones móviles de los trabajadores. Para ello, se apoya en un fichero de configuración local que define los parámetros necesarios para su funcionamiento: el horario de inicio y de fin de la jornada, el intervalo entre escaneos y la lista de identificadores de los empleados autorizados.

El filtrado de anuncios BLE realizado por el *script* permite descartar señales irrelevantes y centrarse exclusivamente en aquellos anuncios que corresponden a los trabajadores de la Fundación Intras, minimizando así la probabilidad de falsos positivos y garantizando la fiabilidad del sistema. Cabe destacar que, en esta primera fase, tanto la aplicación móvil como el *script* de escaneo funcionan de forma independiente, sin una coordinación directa entre ellos. La responsabilidad de configurar correctamente los horarios de emisión recae en el propio trabajador, lo que puede generar cierta dispersión y falta de uniformidad en la gestión de las jornadas laborales.

1.3.2. Segunda fase: Integración mediante servidor MQTT para sincronización

La segunda fase del proyecto abordó la necesidad de integrar de forma centralizada los distintos componentes del sistema, buscando una mayor coordinación y simplificación de las tareas de configuración. Para lograr este objetivo, se implementó un servidor MQTT que actúa como núcleo de comunicación y sincronización entre las aplicaciones móviles de los empleados y los *Gateways* instalados

en los domicilios.

El protocolo MQTT (Message Queuing Telemetry Transport) es una solución ampliamente utilizada en entornos de Internet of Things (IoT) por su ligereza, eficiencia y facilidad de implementación en dispositivos con recursos limitados. Gracias a su arquitectura basada en el modelo publicador-suscriptor, permite enviar mensajes de manera asíncrona y en tiempo real, facilitando la comunicación entre los diferentes elementos del sistema sin necesidad de conexiones persistentes complejas. [4]

Figura 2: Logotipo MQTT [4]

En esta fase, el servidor MQTT centraliza la asignación y publicación de los horarios de trabajo. Por un lado, publica el horario correspondiente a cada aplicación móvil, que permanece conectada y a la espera de recibir esta información. Una vez recibido el horario, la aplicación inicia la emisión de anuncios BLE exclusivamente durante el intervalo de tiempo especificado. Por otro lado, el servidor MQTT también publica la información necesaria para que el script de escaneo en el *Gateway* sepa en qué momentos debe activar la detección de anuncios.

Esta arquitectura permite una sincronización efectiva de las operaciones de emisión y detección, asegurando que el sistema funcione de manera coordinada y precisa. Además, descarga de responsabilidad al trabajador, ya que la configuración de horarios y parámetros se gestiona de forma remota desde el servidor, garantizando la uniformidad de la prestación del servicio y la trazabilidad de las acciones realizadas.

En resumen, esta segunda fase del proyecto representa la consolidación de un sistema robusto, escalable y flexible, en el que la tecnología MQTT desempeña un papel fundamental como herramienta de orquestación de la comunicación y la sincronización de horarios entre todos los componentes involucrados. Esta integración no solo optimiza los recursos tecnológicos, sino que también refuerza la transparencia y la fiabilidad en la gestión de la presencia de los empleados en el marco de la atención domiciliaria que ofrece la Fundación Intras.

1.3.3. Tercera fase: Empleo de una baliza como alternativa a la aplicación móvil

En la tercera y más reciente fase del proyecto, se ha incorporado una nueva solución tecnológica que sustituye la aplicación móvil por una baliza Bluetooth física, específicamente diseñada para emitir anuncios de forma continua y autónoma. Esta baliza se presenta como una alternativa práctica en aquellos escenarios donde el uso de dispositivos móviles no sea viable, no se quiera depender del trabajador o se busque reducir aún más la complejidad operativa del sistema.

La baliza seleccionada emite anuncios en formato *Eddystone*, un estándar definido por Google que permite incluir información personalizada dentro del campo *Service Data* de los paquetes Bluetooth. A través de este campo, se transmiten los identificadores necesarios para su posterior filtrado por parte del *Gateway*, como el ID de empresa y otros valores distintivos del emisor. Este mecanismo mantiene la misma lógica de funcionamiento que la aplicación móvil, asegurando así la compatibilidad con el sistema de escaneo ya implementado.

La incorporación de esta nueva pieza requirió modificaciones significativas en el *script* del *Gateway*, ya que la biblioteca utilizada para escanear los anuncios de las aplicaciones móviles no tenía acceso al campo *Service Data*. El sistema sigue basándose en la comparación de los valores contenidos en este campo con los identificadores configurados previamente, de manera que solo se registran aquellas

detecciones que correspondan a balizas asociadas a trabajadores de la Fundación Intras.

El uso de una baliza Bluetooth presenta varias ventajas frente a la aplicación móvil: mayor autonomía, independencia del trabajador, funcionamiento continuo sin necesidad de interacción, y consumo energético reducido. Además, su instalación en el domicilio del usuario o en el material de trabajo del empleado permite una identificación confiable sin depender de que el dispositivo móvil esté operativo, conectado o configurado adecuadamente.

Esta fase demuestra la flexibilidad y escalabilidad del sistema desarrollado, permitiendo la coexistencia de múltiples tecnologías de emisión sin alterar el núcleo del sistema de detección ni la infraestructura de comunicación basada en MQTT. La posibilidad de alternar entre aplicación móvil y baliza, según las necesidades específicas de cada servicio o usuario, abre la puerta a nuevas formas de implementación y mejora la adaptabilidad del sistema en contextos reales de atención domiciliaria.

1.4. Objetivos del proyecto

El principal objetivo de este proyecto es desarrollar un sistema de registro que permita verificar de forma fiable la presencia de los empleados de la Fundación Intras en los domicilios de los usuarios, utilizando tecnologías móviles, Bluetooth y servidores de mensajería ligera (MQTT). Este sistema debe contribuir a mejorar la transparencia y la calidad de los servicios de atención domiciliaria que ofrece la entidad, optimizando la coordinación y garantizando un registro fiable de la prestación del servicio.

Como objetivos generales, se pretende:

- Implementar una solución tecnológica que permita registrar y verificar la presencia efectiva de los empleados durante el horario de trabajo asignado.
- Integrar las tecnologías Bluetooth y MQTT para conseguir un sistema robusto y escalable, que facilite la coordinación entre la aplicación móvil y el *Gateway* instalado en el domicilio de los usuarios.
- Garantizar la trazabilidad y la fiabilidad de los datos generados, respetando al mismo tiempo la privacidad y la autonomía de los empleados.

Como objetivos específicos, asociados a cada fase del desarrollo, se plantean:

- Desarrollar una aplicación móvil que permita a los empleados de la Fundación Intras introducir sus horarios de trabajo y emitir anuncios BLE exclusivamente durante los periodos configurados.
- Crear un *script* para el *Gateway* instalado en cada domicilio, capaz de escanear de forma selectiva los anuncios BLE de los dispositivos autorizados, filtrando solo aquellos que corresponden a los empleados de la Fundación.
- Diseñar un servidor MQTT que coordine la comunicación entre la aplicación móvil y el Gateway, permitiendo enviar y actualizar horarios de forma remota para sincronizar la emisión y el escaneo de anuncios BLE.
- Conseguir que la aplicación móvil y el *script* en el *Gateway* funcionen de forma sincronizada, evitando falsos positivos y garantizando que el registro de presencia se realice únicamente durante los horarios establecidos.
- Optimizar la gestión de recursos y simplificar la administración del sistema, descargando de responsabilidad a los empleados y centralizando la configuración y supervisión en el servidor.
- Crear un script que escanee el entorno en busca de la baliza Bluetooth y logre registrarla siempre que esté presente.

En conjunto, estos objetivos buscan consolidar una herramienta práctica, fiable y escalable que facilite el control de calidad de los servicios de atención domiciliaria y refuerce el compromiso de la Fundación Intras con la atención personalizada y de calidad a las personas en situación de vulnerabilidad.

1.5. Estructura del documento

A lo largo de este documento se abordarán diversos puntos distribuidos en capítulos, en los que se analizarán diferentes aspectos y fases del proyecto desarrollado. El contenido se estructura en 8 capítulos, que se detallan a continuación:

- **Capítulo 1. Introducción:** En este capítulo se ofrece una breve introducción al caso de estudio, que incluye la contextualización, la definición de conceptos clave, la motivación del proyecto y los objetivos a cumplir.
- **Capítulo 2. Estado del Arte:** Se inicia con un estudio sobre las tecnologías de registro laboral que existen actualmente en el mercado. A continuación, se desarrolla la tecnología Bluetooth que se va a utilizar en el desarrollo del proyecto. Se introducen las versiones del sistema operativo de los dispositivos de la empresa y el lenguaje de programación empleado para el desarrollo de la aplicación. También se habla del *Gateway* localizado en cada lugar de servicio de la empresa y de la tecnología de nube llamado MQTT.
- **Capítulo 3. Estudio de mercado de Dongles Bluetooth:** Se observa un estudio de los adaptadores Bluetooth disponible en el mercado actualmente, evaluando y analizando factores clave a la hora de implementarlo en el proyecto.
- **Capítulo 4. Arquitectura y metodología:** En este capítulo se describe la metodología utilizada para abordar el proyecto, así como las diferentes arquitecturas resultantes del desarrollo modular del mismo.
- **Capítulo 5. Pilotaje de la solución:** Se muestran los resultados obtenidos de la prueba de las aplicaciones y la baliza junto a los escaneos realizados por el *Gateway*, demostrando el correcto funcionamiento del proyecto en todas sus etapas.
- **Capítulo 6. Conclusiones y líneas futuras:** Se exponen las conclusiones extraídas del caso de estudio, junto con propuestas para futuras mejoras de la aplicación y la posibilidad de agregar nuevas funcionalidades.
- **Bibliografía:** Se incluye la bibliografía utilizada tanto para el desarrollo del caso de estudio como para la redacción de este documento, organizada alfabéticamente y siguiendo el formato APA.
- **Anexos:** Este capítulo final contiene los manuales de usuario de las aplicaciones desarrolladas para Android, los *scripts* de Python para el escaneo en el *Gateway* y las dependencias necesarias para poder ejecutarlo

Capítulo 2: ESTADO DEL ARTE

Antes de abordar el desarrollo de la solución propuesta, es imprescindible realizar una revisión del estado actual de las tecnologías relacionadas. Este apartado presenta un análisis de los distintos métodos utilizados actualmente para realizar procesos de registro o control de presencia, incluyendo tanto soluciones tradicionales como propuestas basadas en geolocalización, escaneo de códigos QR, NFC, WiFi y Bluetooth. Se realiza además una comparativa entre ellas para justificar la elección tecnológica adoptada en este proyecto. También se examinan las principales herramientas, lenguajes de programación y plataformas que han sido empleadas en el desarrollo, así como su evolución, características y adecuación al contexto concreto del sistema implementado.

2.1. Tecnologías actuales para el registro laboral

El control de presencia y la verificación del cumplimiento horario son aspectos fundamentales para garantizar la calidad y la eficiencia en múltiples sectores profesionales. Para ello, se emplean diversas tecnologías que permiten registrar la entrada y salida de los trabajadores, cada una con características, ventajas y limitaciones específicas. A continuación, se describen las principales opciones utilizadas en la actualidad para sistemas de registro laboral:

2.1.1. Sistemas basados en geolocalización (GNSS)

El Sistema Global de Navegación por Satélite (GNSS) es un sistema de radionavegación que incluye constelaciones de satélites operadas por distintos países y organizaciones internacionales —como GPS (Estados Unidos), GLONASS (Rusia), Galileo (Unión Europea) o BeiDou (China)— y que ofrece servicios continuos y gratuitos de posicionamiento, navegación y sincronización horaria a usuarios civiles en todo el mundo. Este sistema permite a cualquier persona equipada con un receptor GNSS determinar su ubicación exacta, así como obtener la hora con alta precisión, en cualquier condición meteorológica, de día o de noche, y sin restricciones en cuanto al número de usuarios simultáneos. [5]

Figura 3: Icono Ubicación [Word]

La infraestructura del GNSS se compone de tres segmentos principales: una constelación de satélites que orbitan la Tierra y transmiten señales de radio; una red global de estaciones terrestres responsables del control y monitoreo del sistema; y los receptores GNSS, que procesan las señales recibidas para calcular su posición tridimensional (latitud, longitud y altitud) y la hora local con gran exactitud.

En la actualidad, los receptores GNSS portátiles están ampliamente disponibles en el mercado y permiten a los usuarios conocer su ubicación con alta precisión, facilitando su desplazamiento tanto a pie como en vehículos, embarcaciones o aeronaves. Los sistemas GNSS se han convertido en una herramienta fundamental en todos los modos de transporte, proporcionando soporte esencial en la navegación aérea, marítima y terrestre.

Además de su uso en movilidad, los sistemas GNSS desempeñan un papel clave en múltiples sectores. Los servicios de emergencia y rescate los utilizan para localizar personas y coordinar operaciones en situaciones críticas. También tienen aplicaciones esenciales en infraestructuras como redes

eléctricas, telecomunicaciones y servicios bancarios, donde la precisión temporal resulta crucial. Profesionales como agricultores, topógrafos o geólogos se benefician a diario de las capacidades del GNSS, mejorando la eficiencia, seguridad y precisión de sus tareas gracias a un sistema accesible, fiable y sin coste de uso.

Los sistemas de registro basados en GNSS utilizan la geolocalización por satélite para determinar si un empleado se encuentra en el lugar de trabajo durante un horario establecido. Estos sistemas suelen implementarse mediante aplicaciones móviles que registran la posición geográfica del dispositivo en intervalos determinados.

Entre sus puntos fuertes destacan la posibilidad de verificar la presencia del trabajador en áreas amplias sin necesidad de equipamiento adicional y su adaptabilidad para ser empleados por personal móvil o en entornos exteriores extensos.

En cambio, sus limitaciones residen en la menor precisión en interiores o en zonas urbanas densas debido a la obstrucción de señales por edificaciones, el elevado consumo de batería en los dispositivos móviles y los problemas de privacidad que pueden surgir por la monitorización continua de la ubicación del trabajador.

2.1.2. Sistemas de registro manual

Estos sistemas requieren que el trabajador realice un acto voluntario para marcar su entrada y salida, generalmente mediante escaneos de códigos QR, introducción de contraseñas o uso de tarjetas electrónicas.

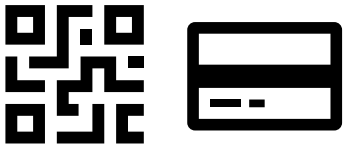

Figura 4: Icono de código QR y tarjeta electrónica

Ofrece una implementación sencilla y de bajo coste, sin necesidad de poseer infraestructura compleja. En cambio, la dependencia de la intervención activa del trabajador es total, pudiendo sufrir olvidos, demoras o fichajes hechos por otras personas.

2.1.3. Sistemas basados en Bluetooth

Bluetooth es un estándar de comunicación inalámbrica de corto alcance que permite el intercambio de datos entre dispositivos electrónicos a través de ondas de radio en la banda de 2.4 GHz. Se caracteriza por su bajo consumo energético (especialmente en su versión BLE), lo que lo hace ideal para aplicaciones móviles, dispositivos portátiles y sistemas de identificación o control de presencia. En el contexto de este último, se utiliza para detectar la proximidad entre el dispositivo del trabajador y un receptor fijo (como un *Gateway*). [6]

Figura 5: Logotipo Bluetooth [6]

Las ventajas de utilizar esta tecnología son: Una alta precisión en entornos interiores, un consumo energético menor en comparación con la tecnología GNSS, la capacidad de automatizar el proceso sin

requerir de una intervención activa del trabajador y la posibilidad de ser integrado en los dispositivos móviles corporativos y sistemas existentes.

Los puntos débiles serían la necesidad de infraestructura adicional para realizar la detección en cada ubicación y la configuración inicial para realizar el filtrado y evitar interferencias.

2.1.4. Sistemas basados en Wi-Fi

Wi-Fi es una tecnología de red inalámbrica que permite la conexión de dispositivos a Internet o a redes locales mediante la transmisión de datos por ondas de radio, normalmente en las bandas de 2.4 GHz y 5 GHz. Utiliza los estándares IEEE 802.11 y es ampliamente utilizada en entornos domésticos, laborales y públicos por su alta velocidad y capacidad de conexión simultánea de múltiples dispositivos. [7]

Figura 6: Logotipo WiFi [7]

El Wi-Fi se utiliza para detectar la presencia mediante la conexión o detección de dispositivos móviles en una red local específica. Algunos sistemas emplean la intensidad de la señal Wi-Fi para estimar la proximidad.

Los puntos fuertes de emplear Wi-Fi serían el aprovechamiento de las infraestructuras de red existentes en muchos lugares de trabajo y un mayor rango que Bluetooth.

En cambio, posee un consumo de energía mayor en dispositivos móviles y tiene una capacidad menor para determinar la ubicación exacta en espacios interiores complejos.

2.1.5. Sistemas basados en NFC (Near Field Communication)

NFC es una tecnología de comunicación inalámbrica de corto alcance (hasta 10 cm) basada en inducción electromagnética que permite el intercambio de información entre dispositivos compatibles. Es ampliamente utilizada para pagos móviles, validaciones de identidad, control de accesos y procesos de registro, gracias a su rapidez, seguridad y facilidad de uso. [8]

Figura 7: Logotipo NFC [8]

La seguridad de esta tecnología es elevada debido a su corto alcance, lo que dificulta la

suplantación. Además, resulta rápido y sencillo para el usuario.

Las desventajas consisten en la necesidad de intervención activa del trabajador para acercar el dispositivo o tarjeta al lector y la imposibilidad de automatizar totalmente el proceso de registro.

2.1.6. Comparativa y justificación de la elección

Tras analizar las tecnologías mencionadas, se concluye que la solución basada en Bluetooth presenta un equilibrio óptimo entre precisión, automatización, consumo energético y facilidad de integración con la infraestructura tecnológica ya disponible en la Fundación Intras.

A diferencia del GNSS, Bluetooth ofrece mayor precisión en interiores y protege mejor la privacidad, ya que no implica un seguimiento constante de la ubicación, sino únicamente la detección en un rango limitado.

Frente a los sistemas manuales, permite la automatización total del proceso, reduciendo errores humanos y fraudes potenciales. Además, se ahorra la instalación de infraestructura adicional.

Aunque requiere la instalación de un *Gateway* con capacidad para escanear anuncios BLE, esta infraestructura ya está presente en los domicilios de los usuarios, lo que facilita la implementación sin costes adicionales significativos.

Comparado con Wi-Fi y NFC, Bluetooth tiene un consumo energético más eficiente que permite un uso sostenido en dispositivos móviles corporativos sin comprometer la autonomía.

Además, la tecnología Bluetooth es compatible con una amplia gama de dispositivos, lo que facilita su despliegue en entornos heterogéneos.

2.2. Tecnología Bluetooth

La tecnología Bluetooth es un estándar de comunicación inalámbrica de corto alcance diseñado originalmente para reemplazar los cables que interconectan dispositivos electrónicos. Desde su primera especificación publicada en 1999 (Bluetooth 1.0), ha evolucionado de manera constante hasta convertirse en una de las tecnologías clave para la comunicación entre dispositivos móviles y el IoT.

En sus primeras versiones (Bluetooth 1.0 y 1.1), Bluetooth ofrecía velocidades modestas de transferencia de datos y un alcance limitado, pero suficiente para aplicaciones de audio y transferencia básica de archivos. Con la llegada de Bluetooth 2.0 + EDR (Enhanced Data Rate), se incrementó el rendimiento y se redujo el consumo energético, lo que permitió su uso más intensivo en dispositivos móviles y periféricos. [9]

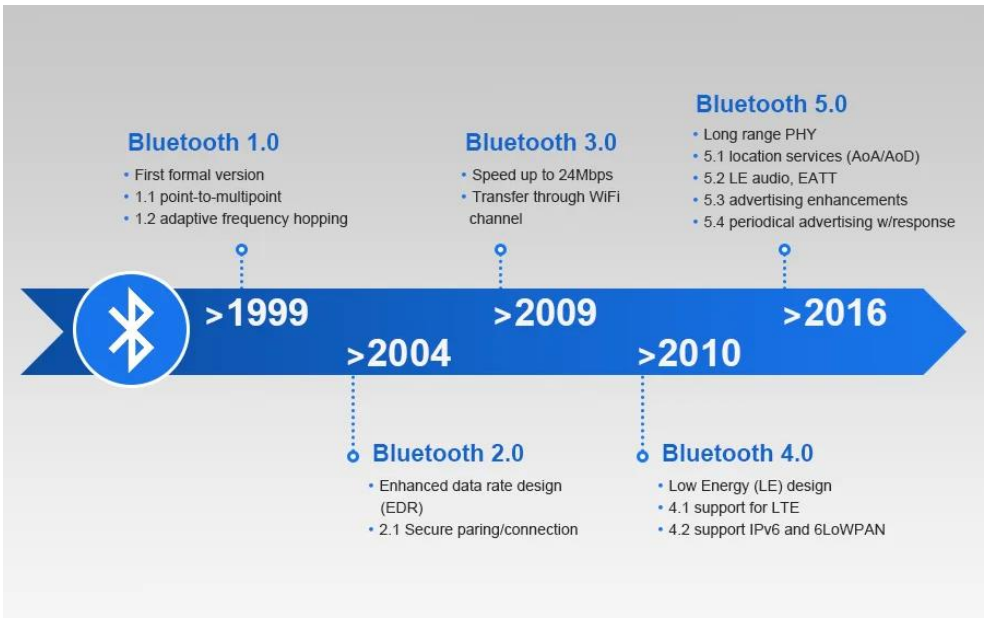

Figura 8: Evolución de la tecnología Bluetooth [9]

Un hito fundamental fue la introducción de BLE en la especificación 4.0 (2010), orientado específicamente a aplicaciones de bajo consumo como dispositivos médicos, *wearables* y sensores IoT. BLE permite mantener comunicaciones periódicas o eventuales con un consumo energético mínimo, ideal para dispositivos que funcionan con baterías y que deben mantener la autonomía durante largos periodos.

En la actualidad, la especificación Bluetooth 5.x (lanzada en 2016 y revisada posteriormente) ha ampliado el alcance (hasta 240 metros en condiciones ideales), ha mejorado la capacidad de transmisión de datos y la robustez frente a interferencias. Estas características hacen de Bluetooth una solución muy adecuada para aplicaciones de verificación de presencia como la que se plantea en este proyecto.

Para este proyecto, al necesitar tener una aplicación corriendo durante horas en un dispositivo móvil realizando anuncios, se fijó como condición mínima el empleo de BLE. Esto no sólo mejoraba la autonomía y el consumo energético, también la potencia de la señal y el rango efectivo.

2.3. Sistema Operativo del teléfono móvil

El sistema operativo Android es la plataforma base sobre la que se construye la aplicación móvil desarrollada en este proyecto. Desde su lanzamiento en 2008, Android se ha consolidado como el sistema operativo dominante en dispositivos móviles, alcanzando más del 70% de la cuota de mercado global según *StatCounter*. Esta adopción masiva ha impulsado el desarrollo de un ecosistema rico en bibliotecas, *frameworks* y herramientas que facilitan la implementación de aplicaciones complejas y el soporte de tecnologías de conectividad como Bluetooth. [10]

Statcounter Global Stats

Mobile Operating Systems	Percentage Market Share
Android	72.72%
iOS	26.92%
Samsung	0.2%
Unknown	0.11%
KaiOS	0.02%
Linux	0.01%

Mobile Operating System Market Share Worldwide - May 2025

Figura 9: Porcentajes de uso de Sistemas Operativos en dispositivos móviles [10]

Android es un sistema operativo de código abierto basado en el *kernel* de Linux, diseñado principalmente para dispositivos móviles con pantalla táctil como *smartphones* y *tablets*. Fue desarrollado inicialmente por Android Inc., y adquirido por Google en 2005. Su arquitectura modular permite una alta personalización por parte de fabricantes y desarrolladores, y su ecosistema está sustentado por el entorno de desarrollo Android Studio, con soporte para lenguajes como Java y Kotlin. Android es el sistema operativo móvil más utilizado del mundo, y su flexibilidad lo hace ideal para soluciones embebidas, aplicaciones personalizadas y sistemas conectados como IoT o dispositivos corporativos. [11]

Figura 10: Logotipo Android [11]

La Fundación Intras ya dispone de una flota de teléfonos móviles corporativos con sistema operativo Android, lo que facilita la integración de la solución sin necesidad de adquirir nuevos dispositivos ni realizar inversiones adicionales. Además, la amplia compatibilidad de Android con las especificaciones de Bluetooth (tanto *Classic* como BLE) permite a la aplicación móvil emitir anuncios BLE de forma eficiente y adaptarse a las diferentes versiones de hardware y software presentes en los dispositivos.

Para garantizar el correcto funcionamiento de la aplicación y aprovechar las últimas mejoras de seguridad y rendimiento, se establece como versión mínima de Android la 12 (API nivel 31). Esta versión incluye importantes avances en la gestión de permisos de Bluetooth y ubicación, así como mejoras en el uso de servicios en segundo plano y notificaciones persistentes. Estas características son esenciales para el correcto funcionamiento de la aplicación de registro, especialmente en un escenario donde la emisión de anuncios BLE y la gestión de tareas en segundo plano son imprescindibles.

Además, Android 12 ofrece un control más granular de los permisos de ubicación y Bluetooth, asegurando que el usuario pueda gestionar de forma sencilla el acceso de la aplicación a estas funcionalidades. Esto resulta clave para garantizar la privacidad y la autonomía de los trabajadores, ya que permite mantener un equilibrio entre funcionalidad y respeto a la intimidad de los profesionales.

2.4. Lenguaje de programación de la aplicación y entorno de desarrollo

Durante la fase inicial del proyecto se valoraron distintas alternativas para el desarrollo de la aplicación móvil, planteando dos enfoques principales: el desarrollo multiplataforma o el desarrollo nativo e independiente para cada sistema operativo.

El desarrollo multiplataforma permite escribir una única base de código que se ejecuta en diferentes sistemas operativos, gracias a la creación de capas o puentes que traducen el código a las plataformas destino. Esta opción ahorra tiempo y recursos, ya que un solo equipo puede mantener un único proyecto y entorno de desarrollo. Sin embargo, la solución multiplataforma a menudo presenta limitaciones en cuanto a acceso a funcionalidades específicas del dispositivo, rendimiento y control fino sobre la experiencia de usuario, además de posibles dependencias en bibliotecas externas que pueden limitar la autonomía del proyecto.

Por otro lado, el desarrollo nativo para el sistema operativo ofrece beneficios claros como mayor independencia tecnológica, mejor rendimiento, acceso completo a las APIs y características propias de cada plataforma, así como una arquitectura más sencilla y cercana al conocimiento de los lenguajes nativos. Esto facilita el control detallado sobre la optimización y la experiencia del usuario.

Finalmente, se optó por el desarrollo nativo, considerando que el proyecto tiene una complejidad moderada y aprovechando los conocimientos adquiridos previamente sobre el lenguaje nativo. Además, esta elección permite un mayor control sobre el rendimiento y la optimización, aspectos críticos para una aplicación que debe gestionar tareas en segundo plano, comunicación BLE y sincronización con un servidor central.

El desarrollo de la aplicación móvil se ha llevado a cabo utilizando Android Studio, el entorno de desarrollo oficial recomendado por Google. Android Studio ofrece integración completa con Kotlin y Java, así como herramientas avanzadas de depuración, perfiles de rendimiento y un emulador de dispositivos para pruebas exhaustivas. Además, incorpora soporte para Gradle, lo que permite una gestión eficiente de dependencias y versiones de bibliotecas. [12]

Figura 11: Logotipo Android Studio [12]

Android Studio también facilita la configuración de permisos y compatibilidades, algo esencial en este proyecto dado que el uso de Bluetooth y la gestión de servicios en segundo plano requieren permisos especiales y comportamientos específicos según la versión del sistema operativo (Android 12 en adelante).

En lo que respecta al desarrollo para Android, los dos lenguajes más utilizados y consolidados son Java y Kotlin. Al contar con experiencia en ambos, se llevó a cabo un análisis comparativo para determinar cuál se adapta mejor tanto a las características del proyecto como a las habilidades del desarrollador.

Java, creado por Sun Microsystems en 1995, es un lenguaje de programación ampliamente consolidado, que ha sido fundamental en la evolución del software moderno. Destaca por ser multiplataforma, orientado a objetos, robusto, con gestión automática de memoria, y contar con una comunidad muy activa y extensa oferta de librerías de código abierto y herramientas como Eclipse. Estas características han hecho de Java un estándar para el desarrollo de aplicaciones, incluyendo Android, durante muchos años. [13]

Figura 12: Logotipo Java [13]

Por su parte, Kotlin es un lenguaje más reciente, desarrollado por JetBrains y presentado en 2011, que busca mejorar la productividad y reducir los errores comunes en programación. Kotlin es oficialmente soportado por Google para Android y ha ganado gran popularidad por su sintaxis más concisa, seguridad

mejorada frente a errores como las referencias nulas, interoperabilidad completa con Java, y soporte tanto para programación orientada a objetos como funcional. Además, dispone de un fuerte respaldo comunitario y de herramientas avanzadas como Android Studio. [14]

Figura 13: Logotipo Kotlin [14]

A continuación, se presenta una tabla comparativa que sintetiza las principales características evaluadas para elegir el lenguaje más adecuado para el desarrollo de la aplicación móvil: [15] [16]

Característica	***Java***	***Kotlin***
Aplicación de ligera extensión		✔
Curva de aprendizaje rápida		✔
Lenguaje simple y sencillo		✔
Potencial futuro del lenguaje	✔	✔
Comunidad y contenido	✔	
Sintaxis concisa		✔

Tabla 1: Comparativa lenguajes Android [Propio]

Finalmente, se decidió utilizar Kotlin, dado que representa el futuro del desarrollo Android, ofrece una sintaxis más limpia y sencilla, y su curva de aprendizaje es más amable, facilitando la rápida incorporación al desarrollo nativo para esta plataforma.

2.5. Sistema Operativo del Gateway

El sistema operativo Debian constituye la base sobre la que se ejecuta el *Gateway* instalado en cada domicilio de los clientes de la Fundación Intras. En este proyecto, se ha utilizado la versión 11 de Debian, que ofrece estabilidad, seguridad y flexibilidad, características esenciales para el entorno de atención domiciliaria en el que se enmarca este sistema.

Debian 11, con nombre en clave *Bullseye*, es la versión número 11 del sistema operativo Debian GNU/Linux, una de las distribuciones Linux más consolidadas y valoradas por su estabilidad, seguridad y filosofía de software libre. Al igual que sus predecesoras, Debian 11 utiliza el *kernel* de Linux y ofrece un extenso repositorio de paquetes de código abierto, lo que permite una configuración altamente flexible para diferentes tipos de entornos, desde servidores hasta sistemas embebidos. Esta versión proporciona soporte a largo plazo (LTS), compatibilidad con múltiples arquitecturas como x86, ARM y PowerPC, y mejoras en el manejo de firmware mediante nuevos paquetes en las secciones *non-free* y *contrib*. Su fiabilidad lo convierte en una elección sólida para infraestructuras críticas, como es el caso del uso en *Gateways* locales dentro de arquitecturas distribuidas orientadas al control y supervisión remota, como el sistema desarrollado en este proyecto. [17]

Figura 14: Logotipo Debian [17]

En la implementación del *Gateway*, Debian 11 se emplea para ejecutar scripts desarrollados en Python, encargados de realizar tareas como el escaneo de anuncios BLE y la conexión al servidor MQTT. Entre los paquetes necesarios se encuentran herramientas como *bluez* (el *stack* Bluetooth oficial para Linux), *python3*, *pip* y *mosquitto-clients* para la gestión de MQTT, además de otros paquetes como cron y systemd que permiten programar y gestionar las tareas en segundo plano de manera eficiente.

El uso de Debian 11 asegura la compatibilidad con el hardware del *Gateway* y garantiza un soporte a largo plazo gracias a las actualizaciones de seguridad y mantenimiento de la comunidad de desarrolladores. Esta elección permite, además, beneficiarse de la amplia documentación y la experiencia acumulada en la implementación de sistemas Linux, reduciendo así la curva de aprendizaje y potenciando la fiabilidad del sistema.

En resumen, Debian 11 constituye la plataforma ideal para el *Gateway* del proyecto, permitiendo la integración fluida de los scripts y las herramientas necesarias para el funcionamiento del sistema de registro y verificación de presencia de los trabajadores.

2.6. Tecnología MQTT

MQTT es un protocolo de mensajería ligero y eficiente, ideal para aplicaciones de IoT y sistemas distribuidos como el desarrollado en este proyecto. Este protocolo sigue un modelo de publicación-suscripción, en el que los dispositivos (clientes) se conectan a un bróker central (servidor) para intercambiar mensajes de manera asincrónica y bidireccional.

En este proyecto, MQTT se utiliza como núcleo de la sincronización entre la aplicación móvil de los trabajadores y el *Gateway* instalado en las viviendas. Esta tecnología permite gestionar de forma centralizada el envío de horarios y parámetros de funcionamiento a ambos dispositivos, optimizando el consumo de recursos y la seguridad de las comunicaciones.

El bróker MQTT actúa como intermediario entre los dispositivos móviles y el *Gateway*. Es el encargado de recibir los mensajes de los dispositivos publicadores y entregarlos a los suscriptores interesados. Para este proyecto, se ha empleado Mosquitto. Mosquitto es un bróker MQTT de código abierto que implementa el protocolo MQTT, diseñado para la comunicación ligera y eficiente entre dispositivos en redes con recursos limitados, como IoT. Debido a su bajo consumo de recursos, alta eficiencia y soporte para múltiples plataformas, Mosquitto es uno de los brókeres MQTT más populares y ampliamente utilizados en proyectos de automatización, domótica y sistemas embebidos. Su arquitectura permite manejar múltiples clientes simultáneamente, gestionando la publicación, suscripción y distribución de mensajes con rapidez y fiabilidad. [18]

Figura 15: Logotipo Mosquitto [18]

Tanto la aplicación móvil como el *script* que se ejecuta en el *Gateway* implementan un cliente MQTT que se conecta al bróker para recibir los mensajes necesarios para activar sus funcionalidades (publicidad BLE o escaneo). En el caso de Android, se utiliza la biblioteca *Eclipse Paho* para Kotlin, que permite integrar un cliente MQTT en las aplicaciones móviles de manera sencilla y eficiente. En el *Gateway*, se utiliza *paho-mqtt* en Python para la suscripción y el procesamiento de mensajes.

En resumen, MQTT se convierte en la tecnología clave para la comunicación entre la aplicación móvil y el *Gateway*, garantizando la sincronización y la eficiencia necesarias para el correcto funcionamiento del sistema de registro.

Capítulo 3: ESTUDIO DE MERCADO DE DONGLES BLUETOOTH

Uno de los componentes clave para el funcionamiento del sistema de registro desarrollado en este proyecto es el adaptador Bluetooth USB (*dongle*), encargado de dotar al Gateway de conectividad Bluetooth para escanear los anuncios BLE emitidos por los dispositivos móviles de los empleados. Debido a que el *Gateway* opera con el sistema operativo Debian 11, no todos los *dongles* del mercado ofrecen compatibilidad nativa, por lo que fue necesario realizar un estudio exhaustivo de los modelos disponibles que se ajustaran a los requisitos técnicos del proyecto.

3.1. Criterios de selección

El proceso de selección se centró en identificar *dongles* Bluetooth que ofrecieran un equilibrio entre compatibilidad, funcionalidad y coste. Los principales criterios de selección considerados fueron:

- Compatibilidad con Debian 11: Se priorizó la compatibilidad con el *kernel* de Linux, especialmente la presencia de soporte oficial o probado en versiones recientes del sistema. Algunos modelos requieren compilar módulos específicos o instalar *firmware* adicional, lo cual puede dificultar el despliegue en gran escala.

- Soporte para BLE: Es imprescindible que el adaptador permita escanear y procesar anuncios BLE, dado que esta es la tecnología utilizada por la aplicación móvil para emitir señales de presencia.

- Rango de alcance: El entorno doméstico puede presentar barreras físicas (paredes, muebles), por lo que se valoraron especialmente los dispositivos que ofrecieran un mayor alcance, garantizando la detección del trabajador desde diferentes estancias de la vivienda.

- Consumo energético: Dado que los *Gateways* pueden estar funcionando continuamente, se optó por dongles de bajo o muy bajo consumo para no afectar el rendimiento general del sistema.

- Facilidad de instalación: Se dio preferencia a los modelos *plug-and-play* o con instalación sencilla, evitando la dependencia de controladores complejos o compilaciones manuales.

- Precio: Como el sistema está destinado a ser replicado en múltiples domicilios, el coste unitario del *dongle* debe mantenerse lo más bajo posible sin comprometer la funcionalidad.

3.2. Modelos analizados

A continuación, se describen brevemente los modelos de *dongles* Bluetooth evaluados durante el estudio:

- Plugable USB 4.0: Un adaptador compacto con soporte oficial para Linux, aunque su disponibilidad es limitada en Europa. Tiene un alcance corto (10 metros) y es fácil de usar, pero carece de soporte explícito para versiones modernas del *kernel*. [19]

Figura 16: Plugable USB [19]

- Kinivo BTD-400: Muy similar al anterior, con soporte para BLE y buena reputación entre usuarios de Linux. Es una opción sólida para entornos sin grandes exigencias de alcance. [20]

Figura 17: Kinivo BTD-400 [20]

- ASUS USB-BT400: Uno de los modelos más populares en su gama, con compatibilidad general con Linux y buen soporte para BLE. Ofrece un alcance de hasta 10 metros y un precio muy competitivo. [21]

Figura 18: ASUS USB-BT 400 [21]

- ASUS USB-BT500: Evolución del modelo anterior, incorpora Bluetooth 5.0, un mayor alcance (hasta 40 metros) y un consumo optimizado. Aunque su compatibilidad con Debian depende del chipset, ha demostrado funcionar correctamente con configuraciones adecuadas. [22]

Figura 19: ASUS USB-BT500 [22]

- Edimax BT-8500: Este modelo fue una de las elecciones finales del proyecto por su alta compatibilidad con el kernel de Linux (desde 2.6.32 hasta 5.3) y soporte para Bluetooth 5.0. Ofrece un alcance de hasta 40 metros y un rendimiento excelente. [23]

Figura 20: Edimax BT-8500 [23]

- Hideez USB: Adaptador compacto con soporte BLE y compatibilidad verificada con varios sistemas operativos. Su alcance (20 metros) y precio lo hacen adecuado para entornos medios, aunque no destaca en ningún aspecto particular. [24]

Figura 21: Hideez USB [24]

- UGREEN USB 5.0: Una opción económica que ofrece buen alcance y muy bajo consumo. Es *plug-and-play* en la mayoría de las distribuciones, aunque su soporte para Debian 11 puede depender del chipset exacto incluido en la revisión del modelo. [25]

Figura 22: UGREEN USB 5.0 [25]

- TP-Link UB500 (con antena externa): El otro modelo seleccionado para el proyecto. Aunque su alcance estándar es de unos 20 metros, su versión con antena externa puede llegar a cubrir entre 40 y 50 metros, lo que lo convierte en una excelente opción para viviendas más amplias o con barreras físicas. Tiene buen soporte con chipsets compatibles en Debian y su instalación es sencilla. [26]

Figura 23: TP-Link UB500 [26]

- Tinxi: Modelo genérico con especificaciones limitadas. Aunque es compatible con Linux y BLE, su procedencia y falta de documentación lo hacen menos recomendable para entornos donde se necesita estabilidad y soporte a largo plazo. Se ha considerado debido a ser un adaptador ya en propiedad. [27]

Figura 24: Tinxi USB [28]

3.3. Elección final

Tras la evaluación técnica y práctica, se optó por utilizar dos modelos de dongles que se ajustan a las necesidades operativas del sistema. El primero es el Edimax BT-8500 por su excelente compatibilidad con Debian y soporte completo para BLE. El segundo es el TP-Link UB500 con antena externa por su mayor alcance, ideal para garantizar la detección del trabajador desde cualquier punto del domicilio. Ambos modelos destacan por su bajo consumo, facilidad de instalación y coste contenido, lo que los convierte en opciones escalables para su implantación en los distintos hogares gestionados por la Fundación Intras.

	Bluetooth	*Linux*	*Precio*	*Rango*	*Consumo*	*Setup*
Plugable USB	4.0	Official Support	No disponible	10m	Bajo	No
Kinivo BTD-400	4.0	Official Support	No disponible	10m	Bajo	Si
ASUS USB-BT400	4.0	Compatible	8€	10m	Bajo	Si
ASUS USB-BT500	5.0	Chipset compatible	13€	40m	Muy bajo	Si
Edimax BT-8500	5.0	Compatible (Kernel 2.6.32~5.3)	19€	40m	Muy bajo	No
Hideez USB	4.0	Compatible	15.50€	20m	Bajo	No
UGREEN USB	5.0	Chipset compatible	9€	18m	Muy bajo	Si
TP-Link UB500	5.3	Chipset compatible	11€	20m	Muy bajo	Si
Tinxi	4.0	Compatible	Propiedad (No disponible)		Bajo	Si

Tabla 2: Comparativa adaptadores Bluetooth [Propio]

Capítulo 4: ARQUITECTURA Y METODOLOGÍA

Este apartado describe la estructura técnica del sistema desarrollado y la metodología seguida para su implementación. Dado que el proyecto involucra múltiples dispositivos y tecnologías (Android, Debian, Bluetooth, MQTT), se optó por una arquitectura modular y una metodología de trabajo iterativa, orientada a construir un sistema funcional, escalable y fácilmente integrable en el contexto real de la Fundación Intras.

4.1. Metodología

Para el desarrollo del sistema se adoptó una metodología iterativa e incremental, adecuada al contexto académico y técnico del proyecto. Esta metodología se caracterizó por el avance por fases funcionales, la validación continua de los resultados y una progresiva integración de los distintos componentes.

Las etapas principales del desarrollo fueron:

- Análisis de requisitos: identificación de necesidades funcionales y técnicas del sistema, así como de las limitaciones del entorno real (Debian 11, dispositivos Android, red local).
- Diseño modular: separación clara de responsabilidades entre la *app* móvil, el *script* del Gateway y el servidor MQTT.
- Implementación progresiva: desarrollo de funcionalidades básicas en cada componente, empezando por el funcionamiento autónomo sin conexión entre dispositivos.
- Pruebas individuales: validación del correcto funcionamiento de la aplicación y el *script* de forma independiente.
- Integración a través de MQTT: conexión de ambos componentes mediante un servidor central y validación conjunta.
- Adaptación del *script* de escaneo para que el *Gateway* sea capaz de detectar y registrar las balizas Bluetooth de la empresa.
- Documentación técnica: recopilación de decisiones, configuraciones y estructura del sistema para facilitar su despliegue y mantenimiento.

Este enfoque permitió alcanzar una solución funcional, fiable y preparada para escalar su uso en entornos reales.

4.2. Objetivos funcionales

El objetivo general de esta versión del sistema es proporcionar una solución funcional de registro que permita registrar de forma automática la presencia de los trabajadores de la Fundación Intras en el domicilio de los usuarios, utilizando tecnologías inalámbricas (BLE) y una arquitectura distribuida basada en dispositivos móviles y *Gateways* con Debian 11. Este sistema debía cumplir los siguientes objetivos funcionales comunes:

- Emitir automáticamente anuncios BLE desde la aplicación móvil durante los periodos de trabajo establecidos.
- Escanear periódica y selectivamente señales BLE por parte del *Gateway*.
- Identificar de forma fiable los anuncios emitidos por dispositivos autorizados (trabajadores).
- Registrar localmente las detecciones válidas como prueba objetiva de presencia.
- Ejecutar ambos componentes en segundo plano, con bajo consumo de recursos.
- Respetar la privacidad y autonomía del trabajador, evitando seguimiento fuera del horario establecido.

Para lograr estos objetivos, se desarrollaron tres versiones funcionales independientes en el proceso de evolución del sistema, cada una con sus propios objetivos particulares.

4.2.1. Arquitectura autónoma

En esta versión, ambos componentes funcionaban sin comunicación entre sí, lo que implicaba que los parámetros debían configurarse manualmente en cada extremo. Los objetivos específicos se dividen entre la aplicación móvil y el script del *Gateway* son los siguientes:

Aplicación móvil:

- Permitir al trabajador introducir el horario manualmente desde la interfaz de la aplicación.
- Configurar los parámetros de envío de publicidad como la frecuencia y el ID para su posterior filtrado en el escaneo.
- Activar la emisión de anuncios BLE únicamente dentro del intervalo definido.
- Mantener la *app* operativa en segundo plano de forma estable.

***Script* en *Gateway*:**

- Leer la configuración de escaneo desde un fichero local.
- Iniciar y detener el escaneo en los horarios programados.
- Detectar anuncios BLE cercanos y filtrar solo los relevantes.
- Registrar las detecciones válidas de forma local.

Esta versión permitió verificar la viabilidad técnica del sistema en condiciones reales, aunque dependía de la correcta configuración manual y ofrecía poca flexibilidad ante cambios o errores.

4.2.2. Arquitectura interconectada

En esta segunda versión, ambos sistemas se conectan a un servidor MQTT central, que actúa como coordinador y fuente de información para sincronizar su comportamiento. Esta arquitectura ofrece mayor automatización, fiabilidad y escalabilidad.

Aplicación móvil (con servidor):

- Conectarse al servidor MQTT y suscribirse al *topic* correspondiente.
- Recibir y almacenar el horario y la frecuencia de emisión desde el servidor.
- Activar automáticamente el servicio de emisión de anuncios BLE durante el intervalo recibido.
- Permanecer en segundo plano sin necesidad de intervención por parte del usuario.

***Script* en *Gateway* (con servidor):**

- Suscribirse al *topic* MQTT asignado para recibir los parámetros de escaneo.
- Activar el escaneo BLE únicamente durante el horario indicado.
- Aplicar filtros para detectar solo los anuncios válidos (empleados autorizados).
- Registrar detecciones de forma estructurada, dejando evidencia de la prestación del servicio.

Esta versión refleja la solución completa y automatizada del sistema, apta para su despliegue en un entorno real, y con posibilidad de extenderse con funcionalidades adicionales como la gestión remota de dispositivos o el envío de registros a la nube.

4.2.3. Arquitectura independiente con la baliza

La tercera versión funcional del sistema introduce una alternativa completamente independiente al uso de dispositivos móviles, mediante la incorporación de una baliza Bluetooth física configurada para emitir de forma autónoma anuncios publicitarios en formato *Eddystone*. Esta versión se diseñó con el objetivo de reducir la dependencia del trabajador y simplificar aún más la operativa del sistema en determinados contextos, como servicios recurrentes o domicilios fijos.

A diferencia de las arquitecturas anteriores, en esta implementación no se requiere intervención del usuario, ni conexión al servidor MQTT por parte de la baliza. Su funcionamiento es continuo, lo que elimina la necesidad de gestionar el encendido del dispositivo, la conectividad o la programación horaria por parte del empleado. Los objetivos funcionales específicos de esta arquitectura fueron los siguientes:

Baliza Bluetooth:

- Emitir de forma constante anuncios BLE sin necesidad de configuración dinámica.
- Incluir en el campo *Service Data* los identificadores necesarios (como el ID de empresa) para que puedan ser reconocidos por el sistema de escaneo.
- Garantizar una alta autonomía energética y estabilidad operativa en cualquier entorno físico.

***Script* en *Gateway*:**

- Leer la configuración de escaneo desde un fichero local.
- Iniciar y detener el escaneo en los horarios programados.
- Detectar los anuncios emitidos por la baliza mediante el filtrado de los datos codificados en el campo *Service Data*.

- Registrar localmente las detecciones válidas, siguiendo el mismo formato que en versiones anteriores, para asegurar la trazabilidad del servicio.

Esta versión ofrece una solución altamente robusta, autónoma y libre de interacción humana, que permite cubrir casos en los que no se disponga de un dispositivo móvil operativo o se prefiera una infraestructura más pasiva. Además, demuestra la flexibilidad del sistema global, capaz de adaptarse a múltiples fuentes de emisión sin modificar la lógica de detección ni comprometer la fiabilidad del registro.

4.3. Arquitectura del sistema

La arquitectura del sistema se desarrolló en dos fases principales. En una primera versión, ambas aplicaciones funcionaban de forma autónoma y sin comunicación entre ellas. Posteriormente, en la segunda versión, se integró un servidor MQTT como elemento central de coordinación, lo que permitió automatizar por completo la sincronización entre componentes.

4.3.1. Arquitectura autónoma

En la primera fase del proyecto, tanto la aplicación móvil como el script del Gateway funcionaban de manera independiente. La *app* permitía al empleado introducir manualmente el horario en el que estaría trabajando en el domicilio, y emitía anuncios BLE durante ese intervalo. Por su parte, el *script* del *Gateway* leía desde un archivo local los parámetros de escaneo (horario, duración, tiempo entre escaneos y lista de identificadores válidos) y escaneaba el entorno en función de esa configuración.

La aplicación se divide en 3 módulos principales, como podemos ver en el diagrama de flujo contenido en la figura 25 (a). El primero consiste en una interfaz de usuario para poder realizar las configuraciones de la frecuencia de envío de anuncios, el ID para su posterior escaneo y un calendario para poder elegir los días y sus horarios de inicio y fin, permitiendo así al trabajador establecer su horario laboral. El segundo módulo se encarga de manejar la lógica de temporización para lanzar cada ciclo, comprobar que estamos en día y horario laboral y que se mantenga funcionando aun estando en segundo plano. El tercer y último módulo está encargado de la gestión, emisión y detención de los anuncios BLE.

En cuanto al *script*, este posee 4 módulos, mostrados en la figura 25 (b). El primero se encarga de la lectura del fichero de configuración local, donde se extraen los parámetros de inicialización como la hora de inicio, la hora de fin, el tiempo entre escaneos y el filtro para el escaneo. El segundo módulo realiza la lógica de temporización para conseguir empezar cada ciclo y comprobar si nos encontramos en horario de escaneo. El tercero permite realizar un escaneo BLE durante unos segundos y filtrar por los dispositivos de interés. El último módulo escribe en un fichero de texto los resultados de cada escaneo, de tal forma que se pueda hacer un seguimiento diario de la actividad de los trabajadores de cada lugar de servicio.

Este enfoque permitió validar el funcionamiento de cada componente por separado, pero requería intervención manual y presentaba riesgo de descoordinación entre la *app* y el escáner.

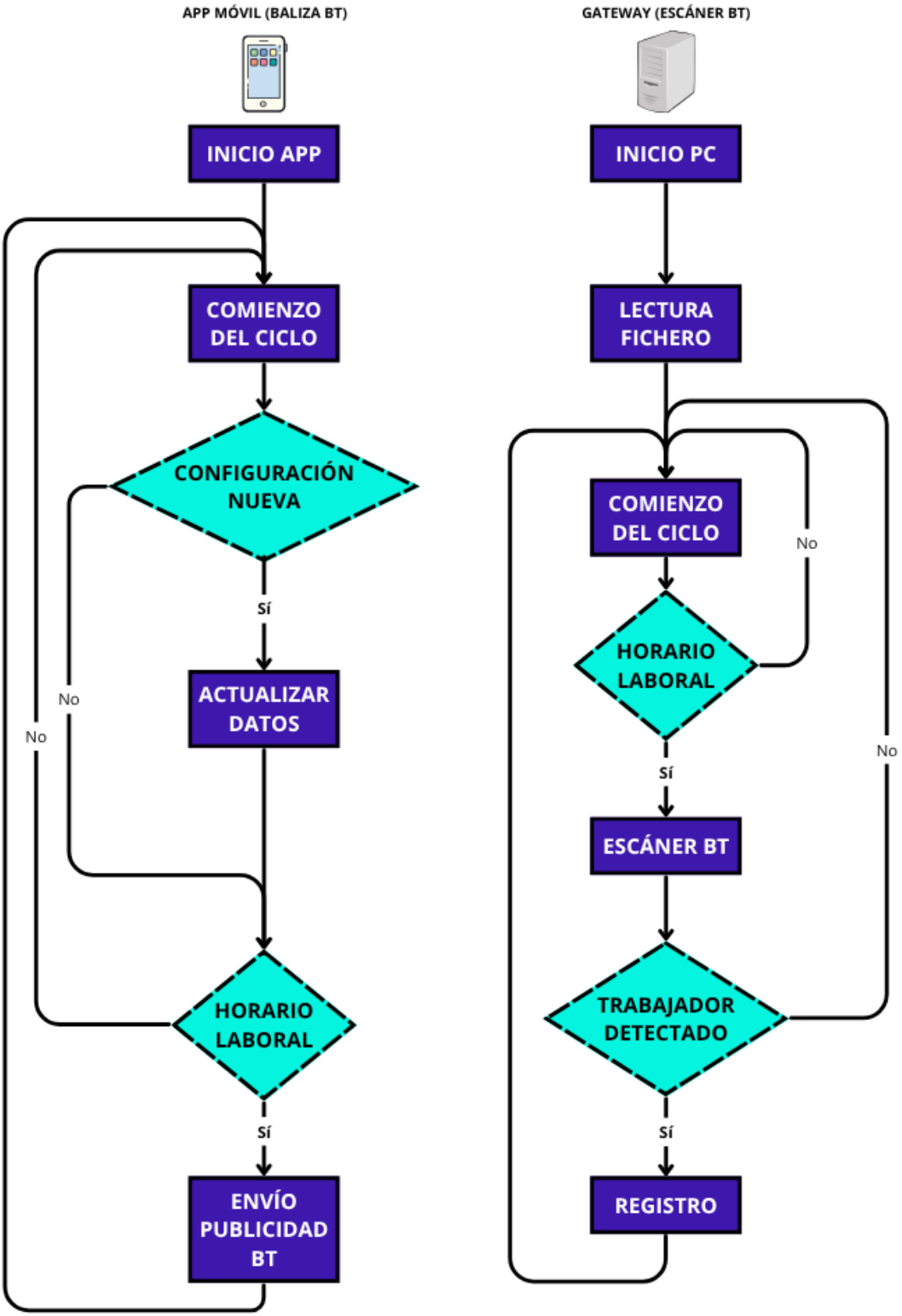

Figura 25: Diagrama de flujo de la aplicación independiente [Propio]

4.3.2. Arquitectura interconectada

En la segunda fase se integró un servidor MQTT como mecanismo de sincronización entre la aplicación móvil y el *Gateway*. Con esta arquitectura, los horarios y parámetros de funcionamiento son definidos por el sistema central y enviados a cada componente de forma automática y personalizada, eliminando la necesidad de configuración manual por parte del trabajador.

La aplicación móvil desarrollada se estructura en cuatro módulos principales, tal como se muestra en el diagrama de flujo representado en la figura 26. El primer módulo corresponde a la interfaz de usuario, que se encarga de mostrar al trabajador la información relevante, así como de almacenar localmente los últimos datos recibidos desde el servidor MQTT (el horario y el ID para el filtro). El segundo módulo implementa la funcionalidad de cliente MQTT, estableciendo la conexión con el servidor y suscribiéndose al topic asignado para recibir los parámetros de funcionamiento. El tercer módulo se ocupa de la gestión de la lógica de temporización, evaluando continuamente si el dispositivo se encuentra dentro del horario laboral previamente configurado. Este módulo también garantiza que el sistema funcione de forma autónoma en segundo plano, incluso cuando la aplicación no está abierta. Finalmente, el cuarto módulo está dedicado al control de la emisión de anuncios BLE, encargándose de iniciar y detener la publicidad en función de la evaluación temporal realizada.

Por su parte, el *script* que se ejecuta en el *Gateway* también se divide en cuatro módulos funcionales, que permiten una operación autónoma y coordinada. El primer módulo implementa el cliente MQTT, que se conecta al servidor y se suscribe al *topic* correspondiente para recibir los parámetros de escaneo (horario de inicio y fin, tiempo entre escaneos y el filtro). El segundo módulo desarrolla la lógica de temporización, encargada de evaluar si el sistema se encuentra dentro del periodo de escaneo activo, y de activar el ciclo de escaneo en los momentos adecuados. El tercer módulo realiza el escaneo BLE, utilizando herramientas del sistema para detectar dispositivos cercanos y aplicar filtros que permitan identificar únicamente los anuncios emitidos por dispositivos válidos (trabajadores autorizados). El cuarto módulo se encarga de la persistencia de datos, registrando en un fichero de texto los resultados de cada sesión de escaneo. Este registro actúa como evidencia objetiva del cumplimiento del servicio, permitiendo un seguimiento diario de la actividad en cada domicilio.

Por último, en esta arquitectura se encuentra el bróker que hará de servidor MQTT. Éste se encargará de realizar la publicación de la configuración de escaneo al *Gateway* y publicará también la configuración de envío de publicidad a la aplicación. De esta forma, se centraliza la lógica de planificación y distribución de tareas, facilitando futuras ampliaciones como el monitoreo remoto o la carga de datos a la nube.

Esta arquitectura descentralizada pero sincronizada permite una gestión eficiente y fiable, con bajo acoplamiento entre los componentes y posibilidad de escalar el sistema a más viviendas sin cambios estructurales.

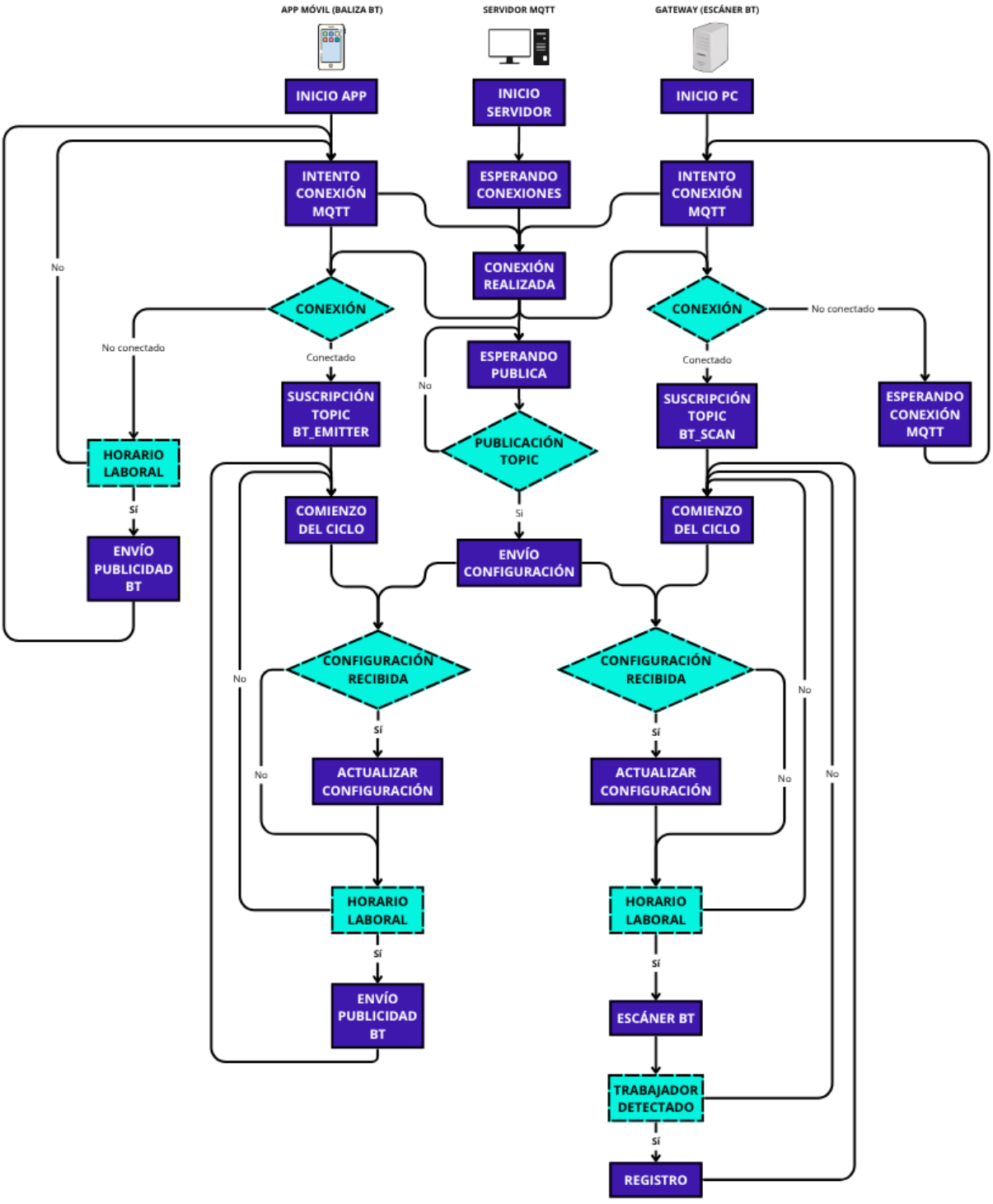

Figura 26: Diagrama de flujo de la aplicación interconectada [Propio]

4.3.3. Arquitectura independiente con la baliza

En esta tercera arquitectura, se plantea una solución en la que se elimina por completo el uso de dispositivos móviles, sustituyéndolos por una baliza Bluetooth física que emite anuncios de forma continua y autónoma. Esta opción busca ofrecer una alternativa aún más simple y pasiva para registrar la presencia de los trabajadores, especialmente útil en entornos donde no se dispone de *smartphones* o se desea evitar depender del usuario final.

A diferencia de la arquitectura interconectada, en este enfoque no existe conexión entre la baliza y el servidor MQTT. La baliza opera de forma completamente autónoma, emitiendo anuncios en formato *Eddystone* a intervalos regulares. Dentro del campo *Service Data* de dichos anuncios se codifican los parámetros necesarios para la identificación del emisor, como el ID de empresa, que será utilizado posteriormente por el *Gateway* para filtrar únicamente aquellas señales válidas y asociadas a trabajadores autorizados.

Desde el punto de vista arquitectónico, el *Gateway* continúa ejecutando el mismo *script* de escaneo que en la versión autónoma inicial. Este lee los parámetros de configuración desde un fichero local, donde se especifican el horario de escaneo, el intervalo entre ciclos, la duración de cada escaneo y el identificador válido. A diferencia del modelo interconectado, no existe sincronización en tiempo real con un servidor central; sin embargo, esto permite implementar la solución de manera rápida, económica y funcional en contextos donde no sea viable establecer conectividad constante con un bróker MQTT.

El script en el Gateway sigue estructurado en cuatro módulos principales, como podemos visualizar en la figura 27. El primero interpreta el fichero que define el periodo de actividad y el ID de filtro. El siguiente módulo determina si el sistema se encuentra dentro del horario definido y lanza los ciclos de escaneo según la frecuencia establecida. El tercer módulo realiza búsquedas en el entorno, accede al campo *Service Data* de los anuncios BLE detectados y aplica los filtros correspondientes. Por último, el cuarto módulo almacena en un fichero de texto todos los anuncios válidos detectados, permitiendo realizar un seguimiento fiable y diario de la actividad registrada.

El uso de esta arquitectura tiene varias ventajas: autonomía total, mayor simplicidad técnica, mínima intervención por parte del trabajador y una reducción en la dependencia de software móvil. Además, permite que el sistema funcione de forma completamente local y sin conexión a Internet, lo que lo hace especialmente adecuado para entornos rurales o con infraestructura limitada.

Aunque esta solución no ofrece la flexibilidad dinámica de la arquitectura interconectada con MQTT, sí mantiene la fiabilidad en la detección de presencia y el principio de funcionamiento basado en la emisión y escaneo de señales BLE. En definitiva, esta arquitectura demuestra la versatilidad del sistema desarrollado, que puede adaptarse a distintos niveles de infraestructura sin perder su funcionalidad principal.

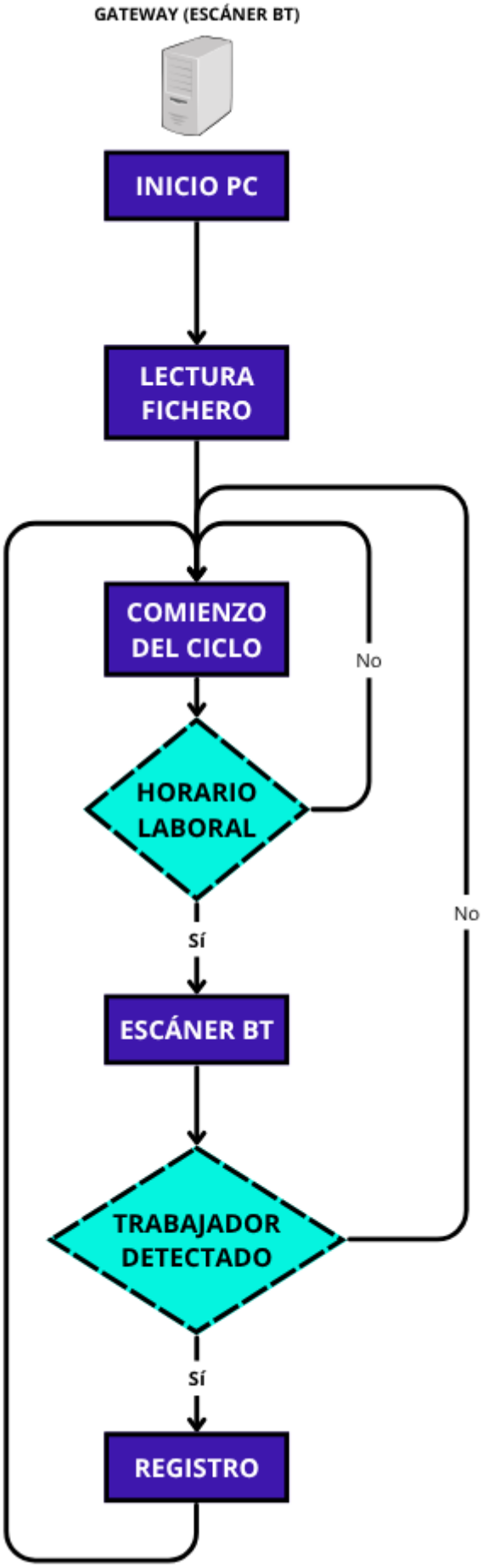

Figura 27: Diagrama de flujo de la arquitectura con baliza [propio]

Capítulo 5: PILOTAJE DE LA SOLUCIÓN

Una vez desarrolladas las distintas versiones funcionales del sistema, se procedió a realizar un proceso de validación experimental que permitiera comprobar su funcionamiento en condiciones reales. Este proceso de pilotaje tuvo como finalidad evaluar la estabilidad, precisión y autonomía de la solución, así como su adaptabilidad a distintos escenarios de uso dentro del contexto de atención domiciliaria ofrecido por la Fundación Intras.

A lo largo de este apartado se presentan las pruebas realizadas sobre cada una de las versiones implementadas —arquitectura autónoma, arquitectura interconectada y arquitectura independiente con baliza— mediante capturas, registros y observaciones técnicas. Para ello, se emplearon dispositivos específicos tanto del lado del emisor (móvil o baliza), como del lado del receptor (*Gateway*), los cuales se describen a continuación.

5.1. Dispositivos utilizados para las pruebas

Con el fin de realizar un pilotaje representativo y reproducible, se seleccionaron dispositivos que reflejaran entornos de despliegue realistas y que garantizaran compatibilidad con las tecnologías involucradas (BLE, sistema operativo Android, Debian GNU/Linux y soporte de controladores actualizados).

5.1.1. Teléfono móvil

Para las pruebas de la aplicación móvil desarrollada en Android, se utilizó un Xiaomi Poco X4 GT, un terminal de gama media-alta con prestaciones suficientes para ejecutar procesos en segundo plano de manera eficiente. Las características técnicas del dispositivo son las siguientes:

- Modelo: Xiaomi Poco X4 GT
- Versión de Android: Android 12 (build SP1A.210812.016)
- Capa de personalización: MIUI 13.0.9 Estable
- Bluetooth: Compatible con BLE
- Procesador: MediaTek Dimensity 8100
- Memoria RAM: 8 GB

Este dispositivo permitió validar tanto el correcto funcionamiento de la aplicación en segundo plano como el envío periódico de anuncios BLE dentro de los horarios configurados. Asimismo, se probó la estabilidad del sistema ante cambios de estado (bloqueo de pantalla, cierre de la aplicación, gestión energética del sistema).

5.1.2. Gateway

Como receptor BLE y punto de procesamiento local, se utilizó un *Gateway* con sistema operativo Debian GNU/Linux 11 (*bullseye*), equipado con la versión del *kernel* adecuada para garantizar la compatibilidad con los adaptadores Bluetooth seleccionados, especialmente el modelo TP-Link con antena externa. La configuración del dispositivo fue la siguiente:

- Sistema operativo: Debian GNU/Linux 11 (bullseye)
- Versión de release: 11
- Kernel: 6.12.22-1~bpo12+1 x86_64 (instalado desde repositorios backports)
- Arquitectura: 64 bits
- Bluetooth: Compatible con adaptadores BLE Edimax y TP-Link (mediante drivers externos)

Este dispositivo fue responsable de ejecutar el *script* de escaneo, conectarse al servidor MQTT (cuando correspondía), y almacenar los registros locales de presencia de forma estructurada.

5.1.3. Baliza

En la tercera fase del proyecto se incorporó el uso de una baliza Bluetooth física como alternativa al teléfono móvil. La baliza utilizada en el pilotaje fue configurada para emitir anuncios en formato *Eddystone*, incluyendo en el campo *Service Data* los identificadores necesarios para su filtrado y detección por parte del *Gateway*.

Las características relevantes de la baliza son:

- Tipo: Baliza BLE configurable
- Formato de emisión: Eddystone (Google)
- Frecuencia de anuncios: Configurable según perfil
- Contenido: ID de empresa codificado en Service Data
- Autonomía: Alta duración gracias a baja frecuencia de emisión
- Compatibilidad: Detectable mediante escaneo BLE con bibliotecas como bluepy

La utilización de esta baliza permitió validar la arquitectura sin móvil, evaluando su detección dentro de distintos contextos y rangos de señal, así como su funcionamiento con distintos adaptadores.

5.2. Resultados de la aplicación independiente

Para verificar el correcto funcionamiento de la aplicación autónoma, se ha realizado una descripción previa de los intervalos de emisión y escaneo empleados. La aplicación móvil emite anuncios BLE durante 10 segundos, seguidos de 50 segundos de espera. Esta configuración se estableció de acuerdo con los requisitos proporcionados por la Fundación Intras, priorizando la eficiencia energética y la mínima intrusión en el dispositivo del trabajador.

En cuanto al *Gateway*, se fijó un ciclo de escaneo de 20 segundos de escucha activa, seguido de 7 segundos de espera. Esta decisión se basó en observaciones experimentales sobre el adaptador Bluetooth TP-Link, el cual posee un buffer interno que descarta anuncios almacenados durante 25 segundos aproximadamente. Para evitar la pérdida de información relevante, se limitó el tiempo de escaneo por debajo de ese umbral. Además, el intervalo entre escaneos se configuró para ser menor que el tiempo total de emisión de la aplicación, garantizando así que el *Gateway* no pierda ningún anuncio mientras se encuentra en estado de espera.

Con esta configuración, se espera que el sistema detecte la presencia del trabajador entre una y dos veces por minuto, en función del solapamiento entre los ciclos de emisión y escaneo. Tal como se observa en la Figura 28, extraída del fichero *scan_results.txt* donde se registran todas las detecciones, el sistema funciona conforme a lo previsto. En cada entrada se muestra la hora en la que finaliza el escaneo, junto con los datos del dispositivo detectado: dirección MAC, RSSI, UUIDs de servicio, y datos de manufactura que contienen el ID de empresa y el ID del dispositivo Android, necesarios para el filtrado.

```
Fecha y hora: 2025-06-13 12:59:49
Dispositivo Encontrado: Dopi - 44:13:25:0A:FC:0A
RSSI: -60 dBm
UUIDs de servicio: ['0000180f-0000-1000-8000-00805f9b34fb']
Manufacturer data: {123: b'a31858b54eef2a47'}
Fecha y hora: 2025-06-13 13:00:45
Dispositivo Encontrado: Dopi - 75:7F:C3:F0:75:24
RSSI: -56 dBm
UUIDs de servicio: ['0000180f-0000-1000-8000-00805f9b34fb']
Manufacturer data: {123: b'a31858b54eef2a47'}
Fecha y hora: 2025-06-13 13:01:40
Dispositivo Encontrado: Dopi - 52:6F:45:4B:18:42
RSSI: -68 dBm
UUIDs de servicio: ['0000180f-0000-1000-8000-00805f9b34fb']
Manufacturer data: {123: b'a31858b54eef2a47'}
Fecha y hora: 2025-06-13 13:02:35
Dispositivo Encontrado: Dopi - 59:77:72:82:05:67
RSSI: -58 dBm
UUIDs de servicio: ['0000180f-0000-1000-8000-00805f9b34fb']
Manufacturer data: {123: b'a31858b54eef2a47'}
Fecha y hora: 2025-06-13 13:03:31
Dispositivo Encontrado: Dopi - 52:7E:C2:F1:88:97
RSSI: -82 dBm
UUIDs de servicio: ['0000180f-0000-1000-8000-00805f9b34fb']
Manufacturer data: {123: b'a31858b54eef2a47'}
```

Figura 28: Resultado escaneo aplicación independiente [propio]

5.3. Resultados de la aplicación interconectada

En esta segunda versión del sistema, los tiempos de emisión, escaneo y espera se mantienen iguales a los definidos en la arquitectura autónoma: 10 segundos de emisión, 50 segundos de descanso, 20 segundos de escaneo, y 7 segundos de espera. El comportamiento esperado sigue siendo detectar entre una y dos presencias por minuto, dependiendo de la alineación temporal de los ciclos.

A diferencia de la versión anterior, esta arquitectura requiere la coordinación inicial mediante un servidor MQTT, que centraliza la configuración horaria de ambos componentes. Como se muestra en la Figura 29, el servidor se inicializa previamente a cualquier acción.

```
PS E:\program files\MQTT\mosquitto> net start mosquitto

El servicio de Mosquitto Broker se ha iniciado correctamente.
```

Figura 29: Inicialización del servidor [propio]

Posteriormente, se publican los datos de configuración necesarios. Primero se envía a la aplicación móvil la información sobre el horario de emisión (por ejemplo, de 11:00 a 14:00) y el ID de empresa correspondiente, a través del *topic bt_emitter*. Después, el *script* del *Gateway* recibe sus parámetros mediante el *topic bt_scan*, incluyendo el horario de escaneo, el intervalo entre ciclos y el identificador para filtrar los anuncios relevantes, tal como se representa en la Figura 30.

```
PS E:\program files\MQTT\mosquitto> .\mosquitto_pub -h 192.168.1.43 -t "bt_emitter" -m "11:00,14:00,123"
PS E:\program files\MQTT\mosquitto> .\mosquitto_pub -h 192.168.1.43 -t "bt_scan" -m "11:00,14:00,7,123"
```

Figura 30: Publicaciones del servidor [propio]

En la Figura 31, se presentan los resultados del fichero de detección. Se observa que el sistema detecta de forma regular un anuncio válido en cada ciclo de escaneo. En una de las ocasiones, el mismo anuncio fue detectado en dos ciclos consecutivos, debido al solapamiento temporal entre los periodos de emisión y escaneo, lo que es coherente con el funcionamiento teórico esperado y reafirma la efectividad del filtrado y la sincronización en esta arquitectura.

```
Fecha y hora: 2025-06-13 13:37:45
Dispositivo Encontrado: Dopi - 43:56:8F:9E:B7:DA
RSSI: -62 dBm
UUIDs de servicio: ['0000180f-0000-1000-8000-00805f9b34fb']
Manufacturer data: {123: b'a31858b54eef2a47'}
Fecha y hora: 2025-06-13 13:38:36
Dispositivo Encontrado: Dopi - 7A:00:94:3E:B3:F7
RSSI: -54 dBm
UUIDs de servicio: ['0000180f-0000-1000-8000-00805f9b34fb']
Manufacturer data: {123: b'a31858b54eef2a47'}
Fecha y hora: 2025-06-13 13:39:01
Dispositivo Encontrado: Dopi - 7A:00:94:3E:B3:F7
RSSI: -52 dBm
UUIDs de servicio: ['0000180f-0000-1000-8000-00805f9b34fb']
Manufacturer data: {123: b'a31858b54eef2a47'}
Fecha y hora: 2025-06-13 13:39:52
Dispositivo Encontrado: Dopi - 4B:83:BF:B3:A7:6D
RSSI: -50 dBm
UUIDs de servicio: ['0000180f-0000-1000-8000-00805f9b34fb']
Manufacturer data: {123: b'a31858b54eef2a47'}
Fecha y hora: 2025-06-13 13:40:43
Dispositivo Encontrado: Dopi - 79:66:2F:40:0C:7A
RSSI: -50 dBm
UUIDs de servicio: ['0000180f-0000-1000-8000-00805f9b34fb']
Manufacturer data: {123: b'a31858b54eef2a47'}
```

Figura 31: Resultado escaneo aplicación interconectada [propio]

5.4. Resultados del escaneo de la baliza

Por último, se evaluó la arquitectura basada en una baliza Bluetooth física, la cual emite anuncios en formato *Eddystone* con una frecuencia aproximada de uno cada 15-20 segundos. Para asegurar la captura de al menos un anuncio por cada ciclo, se configuró el escaneo del Gateway con una duración de 20 segundos, seguida de 10 segundos de espera entre ciclos.

El funcionamiento esperado en esta configuración es que el *Gateway* detecte un anuncio válido en cada escaneo, siempre que la baliza se mantenga dentro del rango de detección.

En la Figura 32, se muestra el contenido del fichero *scan_results.txt* correspondiente a esta prueba. Cada entrada del fichero contiene el instante de finalización del escaneo, la dirección MAC del dispositivo detectado, el RSSI y, especialmente, el campo *Service Data* donde se codifican los datos de identificación necesarios para el filtrado.

Estos datos, definidos por la empresa, incluyen el *namespace* ID y el *instance* ID, configurados previamente en la baliza. Como se puede apreciar en los resultados, cada ciclo de escaneo logra detectar un anuncio emitido por la baliza, lo que confirma el correcto funcionamiento del sistema y la fiabilidad del filtrado en base a la estructura del campo *Service Data*.

```
Fecha y hora: 2025-06-18 18:42:20
Dispositivo Encontrado: ac:23:3f:26:f1:d6
RSSI: -60 dBm
Service Data: aafe00e81234567890123456789ac233f26f1d6
Namespace ID: 12345678901234567890
Instance ID: ac233f26f1d6
----------------------------
Fecha y hora: 2025-06-18 18:42:50
Dispositivo Encontrado: ac:23:3f:26:f1:d6
RSSI: -54 dBm
Service Data: aafe00e81234567890123456789ac233f26f1d6
Namespace ID: 12345678901234567890
Instance ID: ac233f26f1d6
----------------------------
Fecha y hora: 2025-06-18 18:43:21
Dispositivo Encontrado: ac:23:3f:26:f1:d6
RSSI: -60 dBm
Service Data: aafe00e81234567890123456789ac233f26f1d6
Namespace ID: 12345678901234567890
Instance ID: ac233f26f1d6
----------------------------
Fecha y hora: 2025-06-18 18:43:51
Dispositivo Encontrado: ac:23:3f:26:f1:d6
RSSI: -54 dBm
Service Data: aafe00e81234567890123456789ac233f26f1d6
Namespace ID: 12345678901234567890
Instance ID: ac233f26f1d6
----------------------------
Fecha y hora: 2025-06-18 18:44:21
Dispositivo Encontrado: ac:23:3f:26:f1:d6
RSSI: -60 dBm
Service Data: aafe00e81234567890123456789ac233f26f1d6
Namespace ID: 12345678901234567890
Instance ID: ac233f26f1d6
----------------------------
Fecha y hora: 2025-06-18 18:44:51
Dispositivo Encontrado: ac:23:3f:26:f1:d6
RSSI: -60 dBm
Service Data: aafe00e81234567890123456789ac233f26f1d6
Namespace ID: 12345678901234567890
Instance ID: ac233f26f1d6
----------------------------
Fecha y hora: 2025-06-18 18:45:21
Dispositivo Encontrado: ac:23:3f:26:f1:d6
RSSI: -52 dBm
Service Data: aafe00e81234567890123456789ac233f26f1d6
Namespace ID: 12345678901234567890
Instance ID: ac233f26f1d6
----------------------------
```

Figura 32: Resultados del escaneo de la baliza [propio]

Capítulo 6: CONCLUSIONES Y LÍNEAS FUTURAS

6.1. Conclusiones

La realización de este TFM ha permitido abordar el diseño, implementación y validación de un sistema funcional, modular y automatizado de verificación de presencia mediante tecnología BLE, orientado a mejorar la supervisión y la eficiencia de los servicios de atención domiciliaria proporcionados por la Fundación Intras. Durante el desarrollo del proyecto se han afrontado desafíos técnicos, organizativos y éticos, lo que ha contribuido a consolidar conocimientos en programación, comunicaciones distribuidas, configuración de hardware en Linux y diseño de arquitecturas IoT.

Uno de los logros más significativos ha sido la construcción de un sistema descentralizado, escalable y fiable, capaz de detectar de forma autónoma la presencia de los trabajadores en los domicilios de los usuarios, eliminando la necesidad de registros manuales. Este sistema se basa en el uso de teléfonos móviles Android con capacidad BLE, *Gateways* locales con Debian 11, adaptadores Bluetooth compatibles, y un servidor MQTT encargado de sincronizar la emisión y el escaneo en tiempo real. Su diseño modular facilita tanto el despliegue inicial como su futura ampliación o adaptación a otros contextos.

Una de las incorporaciones más destacadas ha sido la introducción de una tercera fase del proyecto, basada en el uso de balizas Bluetooth físicas como alternativa a la aplicación móvil. Estas balizas, configuradas para emitir anuncios en formato *Eddystone*, permiten identificar de forma pasiva y precisa la presencia de un trabajador, sin requerir su intervención ni depender del estado del dispositivo móvil. Esta solución añade una capa extra de flexibilidad y robustez, especialmente útil en escenarios donde se desee minimizar el mantenimiento técnico, aumentar la autonomía operativa o reducir costes relacionados con dispositivos móviles.

Desde el punto de vista técnico, el sistema ha sido estructurado de forma modular y coherente. Cada componente —aplicación móvil, *script* de escaneo en el *Gateway*, baliza BLE y servidor MQTT— ha sido desarrollado para funcionar de forma autónoma pero integrada, facilitando la mantenibilidad, el testeo y la implementación progresiva. La elección de Kotlin para Android y Python junto a la biblioteca *bleak* o *bluepy* para los scripts en Linux ha resultado acertada, permitiendo un desarrollo eficiente con buen soporte documental y comunitario. También se ha realizado un estudio técnico y práctico sobre la compatibilidad de adaptadores Bluetooth, seleccionando modelos que aseguren el correcto funcionamiento bajo Debian, incluyendo tanto dispositivos compactos (Edimax) como de largo alcance (TP-Link con antena externa), garantizando la aplicabilidad en entornos reales.

Paralelamente, se ha tenido en cuenta el respeto a la privacidad y la autonomía del trabajador, procurando que el sistema funcione exclusivamente durante los horarios laborales establecidos, sin capturar datos sensibles ni realizar rastreo fuera de jornada. Esta orientación ética ha sido central en el diseño de la solución, garantizando que el sistema no solo sea técnicamente viable, sino también socialmente aceptable.

En definitiva, este trabajo ha dado lugar a una solución innovadora, funcional y adaptable, capaz de resolver una necesidad concreta dentro del ámbito asistencial. Ha sido desarrollada bajo criterios de eficiencia, bajo coste, privacidad, modularidad y facilidad de despliegue, y queda preparada para su integración futura con otros sistemas o servicios. La experiencia adquirida ha permitido no solo profundizar en conocimientos técnicos avanzados, sino también comprender mejor el impacto social y humano de la tecnología aplicada a contextos sensibles como el cuidado de personas dependientes.

6.2. Líneas futuras

Aunque el sistema desarrollado ha alcanzado una versión funcional, validada y aplicable en escenarios reales, existen diversas líneas de trabajo que podrían explorarse para mejorar, escalar o diversificar sus funcionalidades. A continuación, se detallan algunas propuestas relevantes:

- Cifrado y seguridad en las comunicaciones: Aunque el uso de MQTT en redes locales simplifica la configuración, en un despliegue a gran escala sería recomendable aplicar medidas adicionales de seguridad, como el cifrado TLS y autenticación mediante certificados, tanto en el bróker como en los dispositivos.

- Comunicación bidireccional y registro remoto: Actualmente, los datos de detección se almacenan localmente en el *Gateway*. Como evolución natural, se podría implementar un sistema de envío automático de estos registros a un servidor central o base de datos, lo que permitiría supervisar en tiempo real la actividad de los empleados desde una interfaz web o panel de control.

- Actualización dinámica de dispositivos: El sistema actual requiere reinicios manuales o cambios en ficheros locales para modificar configuraciones. Una mejora sería permitir actualizaciones OTA (Over The Air) o a través de comandos MQTT adicionales que permitan modificar el comportamiento del script o la aplicación sin intervención física.

- Compatibilidad multiplataforma: Aunque se optó por desarrollo nativo en Android, una versión equivalente para iOS permitiría ampliar el sistema a dispositivos Apple, en caso de que en el futuro se empleen terminales mixtos en la Fundación.

- Interfaz de gestión para coordinadores: Actualmente la configuración horaria se realiza de forma manual o mediante envío desde el servidor. Un posible desarrollo sería la creación de una interfaz web para el personal de gestión, desde la cual se puedan asignar horarios, monitorizar presencia y consultar reportes, todo centralizado.

- Escalabilidad e instalación automatizada: Para facilitar la implementación del sistema en decenas o cientos de domicilios, podrían desarrollarse *scripts* automatizados que configuren los *Gateways* desde cero, incluyendo la instalación del sistema operativo, los *drivers*, el *script* de escaneo y la configuración del cliente MQTT.

- Optimización energética y rendimiento: Aunque la aplicación móvil ya dispone de distintos modos de emisión (bajo consumo, equilibrado, baja latencia), sería interesante analizar más profundamente el impacto real de cada uno en distintos dispositivos y explorar opciones de ajuste dinámico según batería o conectividad.

- Integración con balizas inteligentes o configurables: A partir de la experiencia adquirida con la implementación de la baliza BLE, se podrían explorar modelos que permitan configuración remota, gestión centralizada o respuesta activa a comandos, integrando así funcionalidades más avanzadas.

Estas propuestas ofrecen una base sólida para continuar la evolución del sistema en futuros desarrollos, ya sea como producto real desplegado en entornos asistenciales, como proyecto de investigación aplicada, o como plataforma base para soluciones IoT centradas en personas.

BIBLIOGRAFÍA

[1] **Intras**. (s. f.). https://www.intras.es/

[2] **Reglamento (UE) 2016/679 del Parlamento Europeo y del Consejo (RGPD)**. (2016). https://eur-lex.europa.eu/legal-content/ES/TXT/?uri=CELEX%3A32016R0679

[3] **Boletín Oficial del Estado (LO 3/2018)**. (2018). https://www.boe.es/eli/es/lo/2018/12/05/3

[4] **OASIS**. (s. f.). MQTT Version 3.1.1 Specification. https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

[5] **GNSS and Secure SATCOM User Technology Report. (s. f.).** EU Agency For The Space Programme**. https://www.euspa.europa.eu/tech-report**

[6] **Bluetooth SIG**. (s. f.). Bluetooth Core Specification Version 6.0. https://www.bluetooth.com/specifications/specs/core-specification-6-0/

[7] **IEEE Standards Association**. (s. f.). IEEE 802.11. https://standards.ieee.org/ieee/802.11/7131/

[8] Coskun, V., Ok, K., & Ozdenizci, B. (2015). *Near Field Communication (NFC): From Theory to Practice*. Wiley.

[9] **Mokosmart**. (s. f.). Guide on different Bluetooth versions. https://www.mokosmart.com/es/guide-on-different-bluetooth-versions/

[10] **StatCounter Global Stats**. (s. f.). OS Market Share Mobile Worldwide. https://gs.statcounter.com/os-market-share/mobile/worldwide

[11] **TechTarget**. (s. f.). Android OS definition. https://www.techtarget.com/searchmobilecomputing/definition/Android-OS

[12] **Android Developers**. (s. f.). Android Studio Introduction. https://developer.android.com/studio/intro?hl=es-419#:~:text=Android%20Studio%20es%20el%20entorno,desarrollo%20de%20apps%20para%20Android.

[13] ***Java***. *(s. f.). What is Java? https://www.java.com/es/download/help/whatis_java.html*

[14] ***hack(io)***. *(s. f.). Kotlin: Aprende todo sobre el mejor el lenguaje para aplicaciones móviles en Android. https://www.hackio.com/blog/kotlin-aprende-todo*

[15] Galindo, S. (2022, 29 septiembre). Ventajas del desarrollo de aplicaciones en lenguaje JAVA. *3digits*. https://www.3digits.es/blog/ventajas-lenguaje-java.html#:~:text=La%20tecnolog%C3%ADa%20Java%20permite%20acceder,como%20Windows%2C%20Mac%20y%20Linux.

[16] Domínguez, S. (2023, 21 diciembre). Kotlin vs Java: Comparativa en profundidad. *OpenWebinars.net*. https://openwebinars.net/blog/kotlin-vs-java/

[17] **Debian Project**. (s. f.). Debian 11 "Bullseye" Release Notes. https://www.debian.org/releases/bullseye/amd64/release-notes/index.es.html

[18] **Eclipse Foundation**. (s. f.). *Eclipse Mosquitto - An open source MQTT broker*. https://mosquitto.org/

[18] **Plugable**. (s. f.). USB-BT4LE Product. https://plugable.com/products/usb-bt4le

[19] **Kinivo**. (s. f.). BTD-400 User Manual. http://downloads.kinivo.com/product/manual/BTD-400%20User%20Manual.pdf

[20] **ASUS**. (s. f.-a). USB-BT400 Adapter. https://www.asus.com/es/networking-iot-servers/adapters/all-series/usbbt400/

[21] **ASUS**. (s. f.-b). USB-BT500 Adapter. https://www.asus.com/es/networking-iot-servers/adapters/all-series/usb-bt500/

[22] **Edimax**. (s. f.). BT-8500 Bluetooth Adapter. https://www.edimax.com/edimax/merchandise/merchandise_detail/data/edimax/es/bluetooth/bt-8500/

[23] **Hideez**. (s. f.). Hideez Dongle. https://hideez.com/es-es/products/hideez-dongle?srsltid=AfmBOoqpKSzW3IIJ7nBqdC6kC9_Gv-l41MCpYYZ5iHsqE-VqYxr0QndF

[24] **UGREEN**. (s. f.). Bluetooth Adapter Product Page. https://eu.ugreen.com/products/80889

[25] **TP-Link**. (s. f.). UB500 Plus Adapter. https://www.tp-link.com/uk/home-networking/adapter/ub500-plus/

[26] **Tinxi**. (s. f.). Bluetooth Adaptador palillo. https://www.amazon.es/tinxi%C2%AE-Bluetooth-Adaptador-palillo-velocidad/dp/B01F75JR42

ANEXO I: MANUAL DE USUARIO DE APLICACIÓN AUTÓNOMA

Al iniciar la aplicación por primera vez, se mostrará una ventana emergente que redirige al usuario a los ajustes de batería del sistema, con el objetivo de excluir la aplicación de las restricciones de optimización. Esta configuración es esencial, ya que la aplicación está diseñada para funcionar de forma continua en segundo plano. Si el usuario aplica correctamente esta configuración, la advertencia no volverá a aparecer en futuros inicios. (Figura 33)

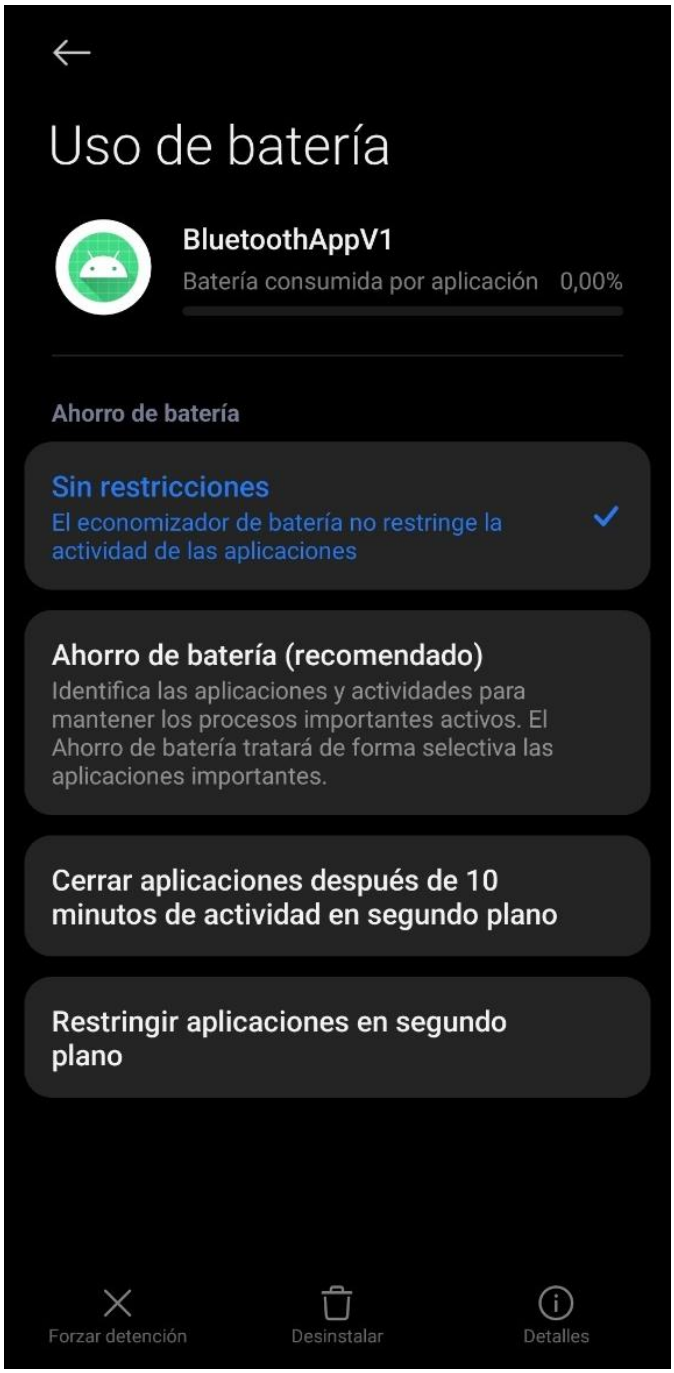

Figura 33: Configuración batería [Propio]

Acto seguido, la aplicación solicita los permisos necesarios para su funcionamiento. En primer lugar, se pide al trabajador que conceda el permiso de ubicación, indispensable para el uso correcto de BLE, según las políticas de Android. Si el usuario acepta este permiso, no se le volverá a solicitar mientras permanezca activo. Esto viene reflejado en la figura 34.

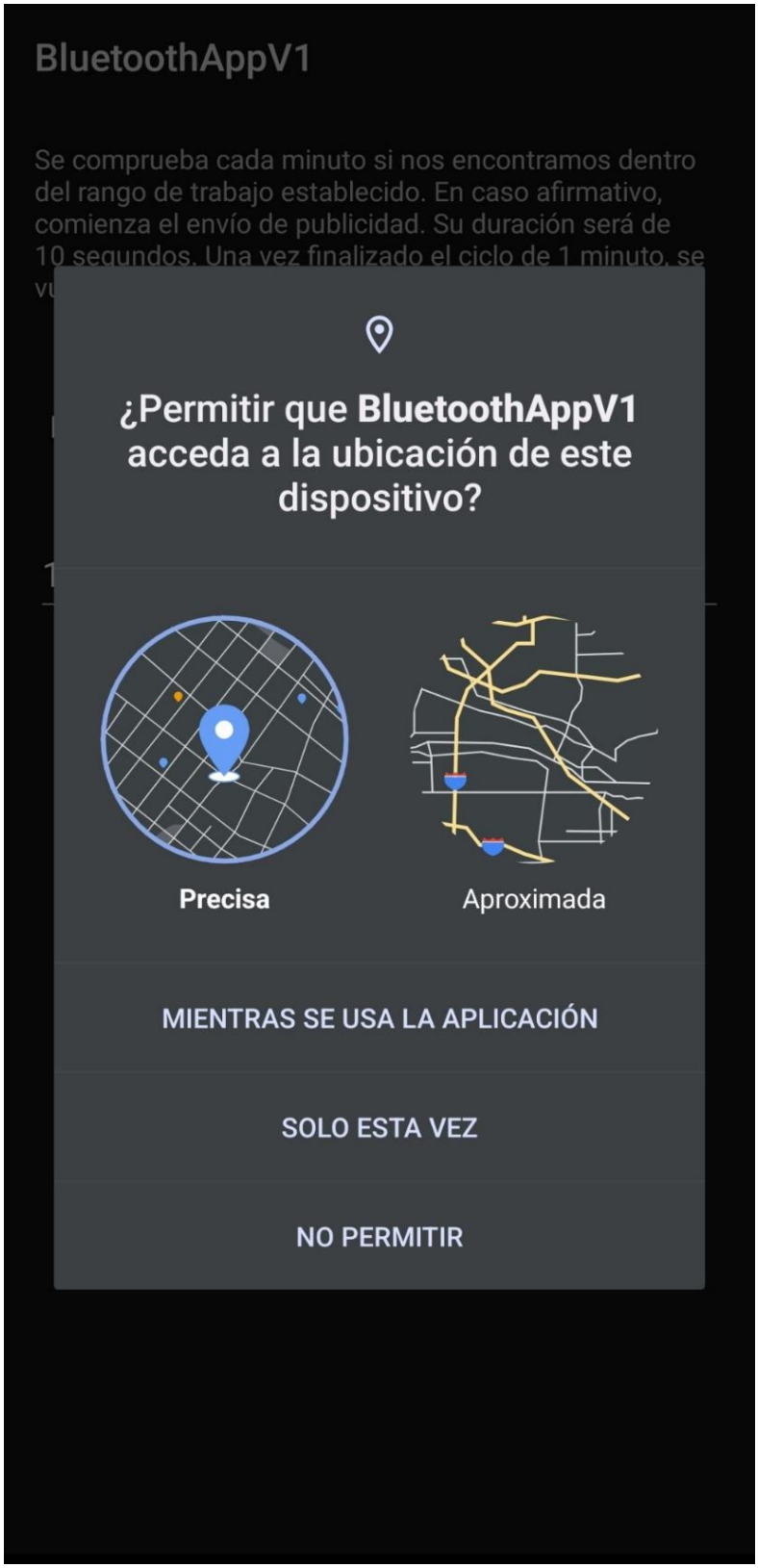

Figura 34: Permiso Ubicación [Propio]

A continuación, se solicita el permiso de acceso a Bluetooth, esencial para iniciar cualquier operación relacionada con esta tecnología. Sin él, la aplicación no puede realizar la emisión de anuncios. Del mismo modo, esta ventana no volverá a mostrarse si el permiso se mantiene concedido. Dicha ventana la podemos ver en la figura 35.

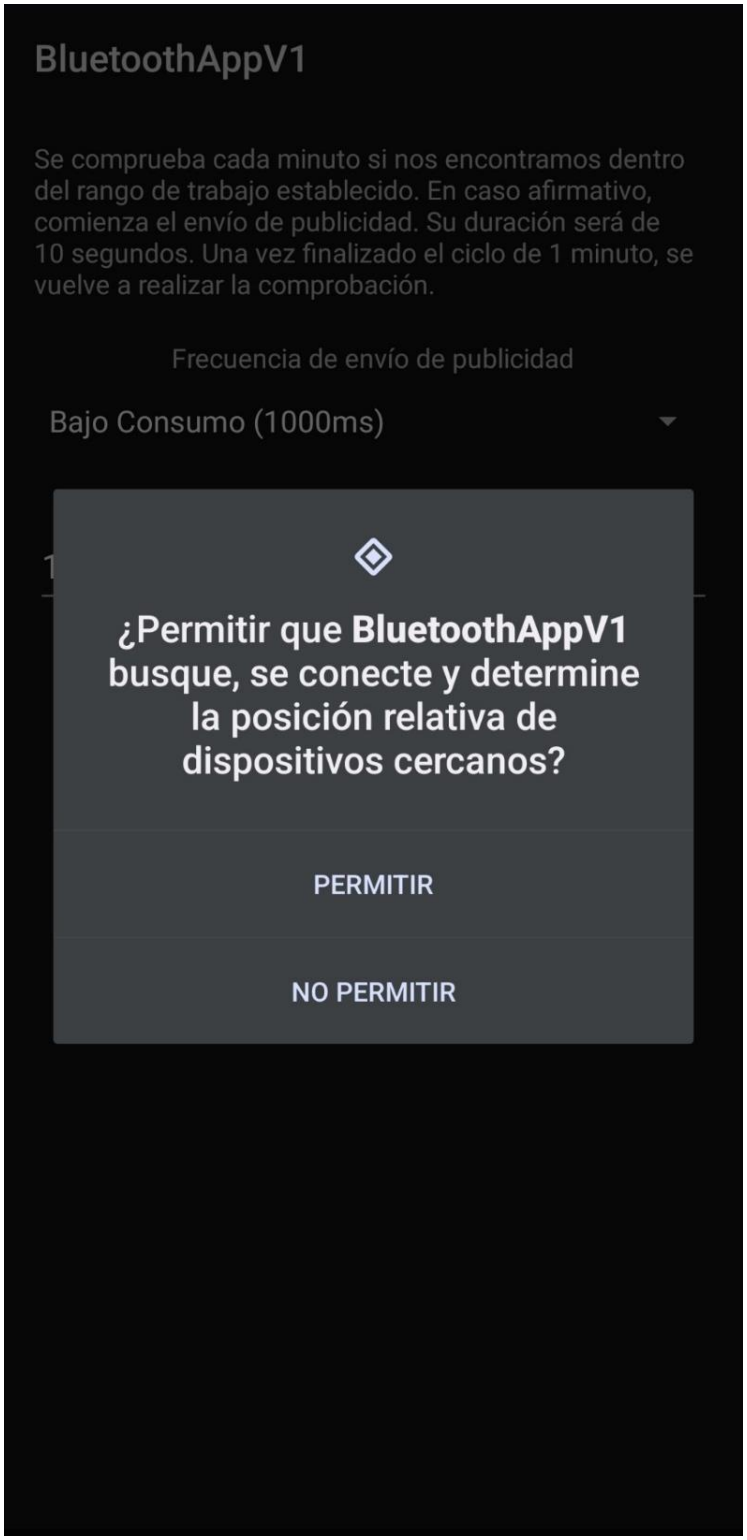

Figura 35: Permiso Bluetooth [Propio]

Lo siguiente que realizará la aplicación es comprobar si están habilitados tanto la ubicación como el Bluetooth. Esto es vital para el funcionamiento de esta. Como vemos en la figura 36, si la ubicación está desactivada nos llevará directamente a la configuración del dispositivo, donde tendremos que activarla para poder utilizarla correctamente.

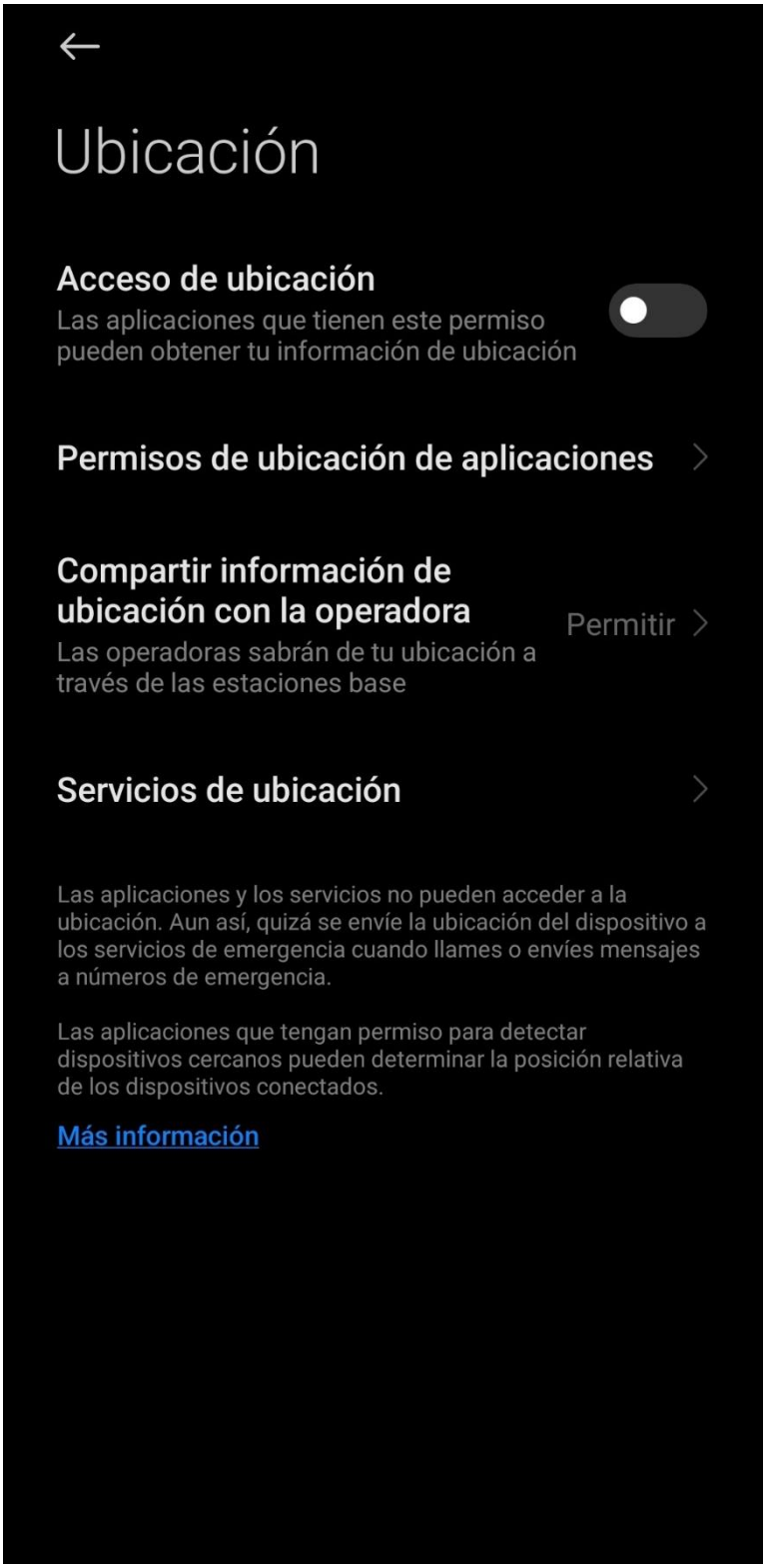

Figura 36: Activación de ubicación [Propio]

La siguiente ventana emergente (figura 37) saldrá siempre que esté deshabilitado el Bluetooth en el dispositivo. Para que la aplicación pueda realizar su función, debemos pulsar en permitir para poder activar el Bluetooth y realizar los envíos de publicidad.

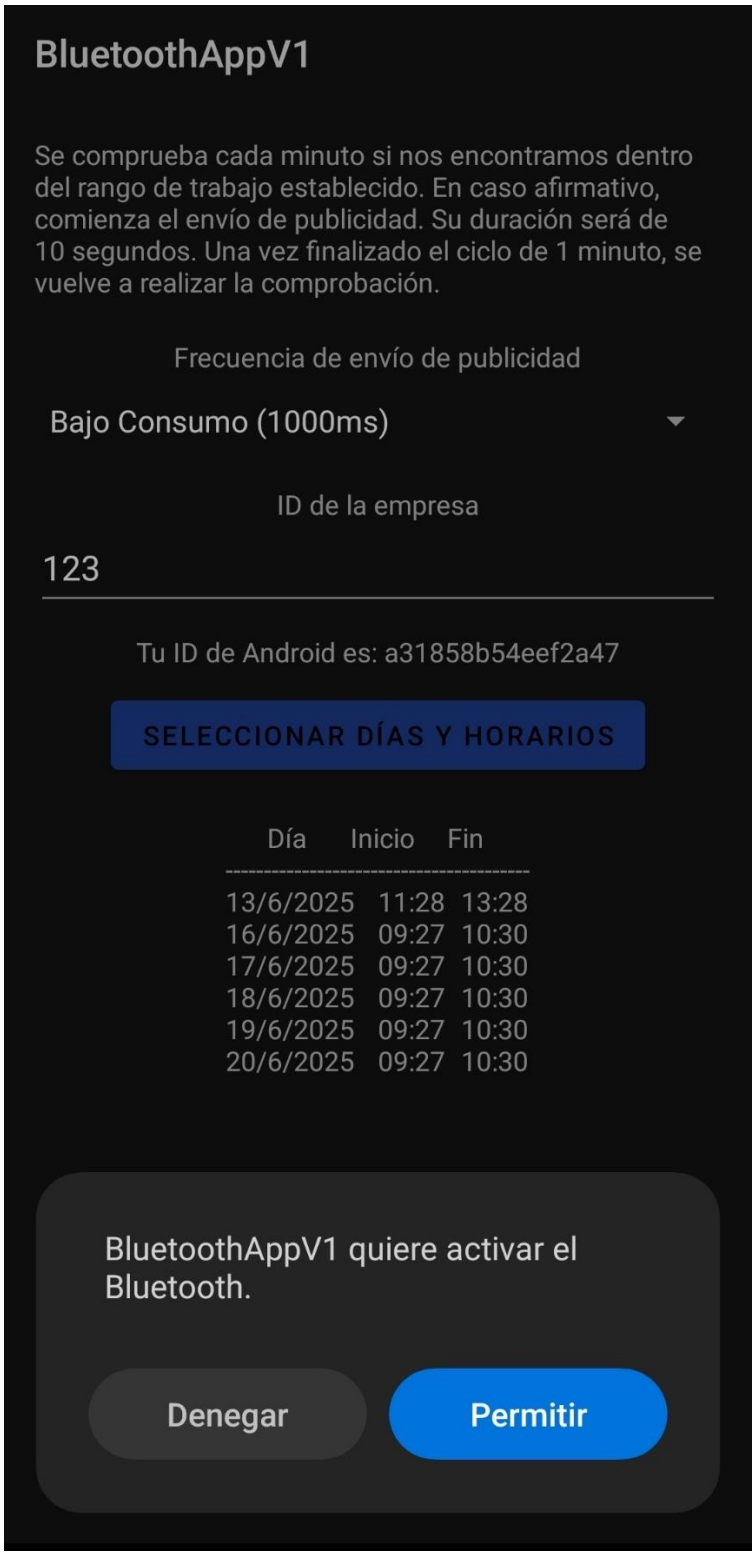

Figura 37: Activación Bluetooth [Propio]

Una vez realizada la concesión de permisos, la configuración de batería y las habilitaciones pertinentes, tenemos la vista principal de la aplicación, como podemos observar en la figura 38. En ella, observamos varios elementos que explicaremos de forma individual a continuación.

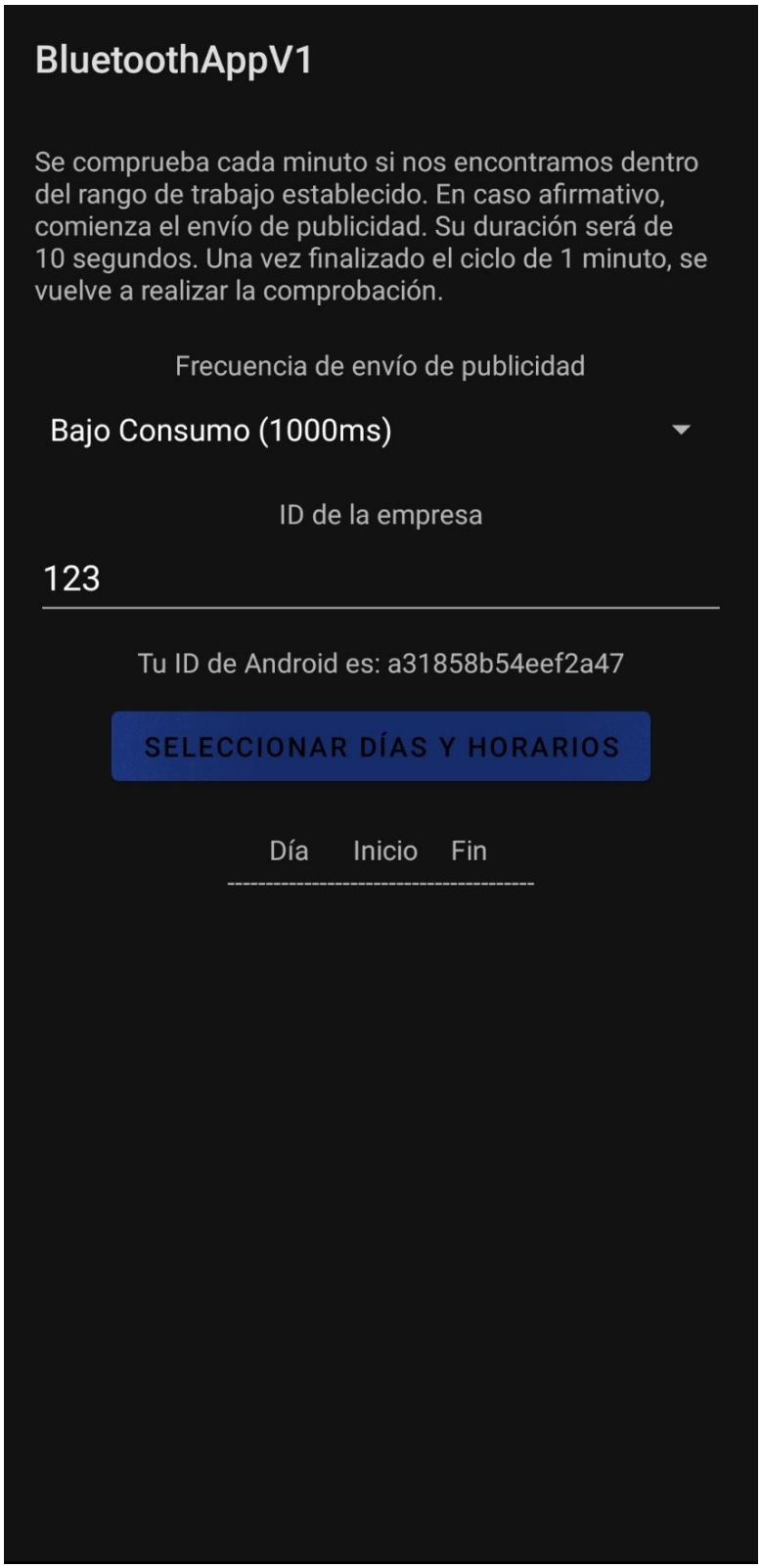

Figura 38: Vista principal de la aplicación [Propio]

En la figura 39 se encuentra un texto explicativo que describe el funcionamiento del sistema. La aplicación ejecuta ciclos periódicos de 60 segundos. Durante cada ciclo, comprueba si el usuario se encuentra en el horario laboral previamente configurado. En caso afirmativo, comienza la emisión de anuncios BLE durante un periodo de 10 segundos. Una vez transcurrido el minuto completo desde el inicio del ciclo, el proceso se repite automáticamente.

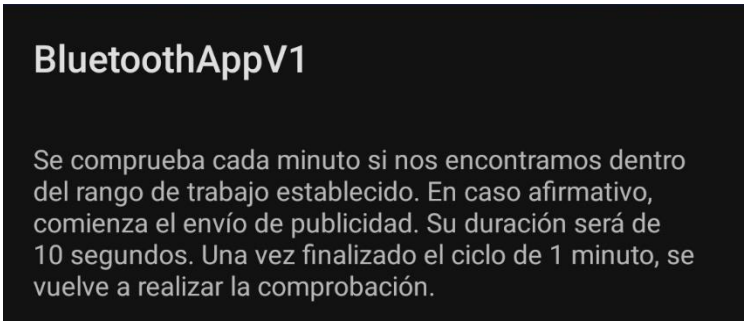

Figura 39: Texto informativo de la app [Propio]

Dentro de la misma vista, el usuario puede configurar la frecuencia con la que se enviarán los anuncios publicitarios. Como vemos en la figura 40, existen tres modos de funcionamiento. El modo Bajo Consumo, con una frecuencia de 1000 ms, permite emitir aproximadamente 10 anuncios por ciclo y es el que menos batería consume. El modo Equilibrado, con una frecuencia de 250 ms, genera unos 40 anuncios por ciclo y representa una opción intermedia entre consumo y visibilidad. Por último, el modo Baja Latencia, con una frecuencia de 100 ms, permite enviar hasta 100 anuncios en cada ciclo, siendo el más intensivo en consumo energético, pero también el más eficaz en cuanto a número de emisiones.

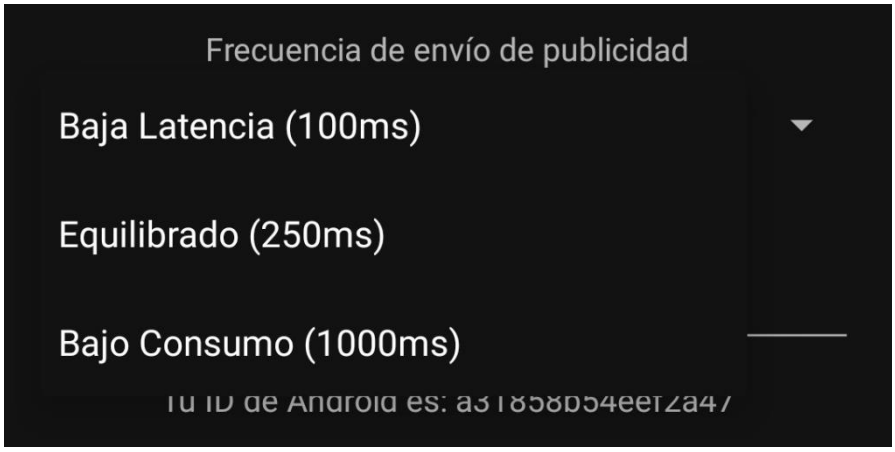

Figura 40: Configuración frecuencia de publicidad [Propio]

A continuación, el usuario puede configurar el identificador de empresa. Este ID será incluido en cada anuncio BLE y servirá como criterio de filtrado en el Gateway, permitiendo asociar las señales con los empleados de una misma organización. (Figura 41)

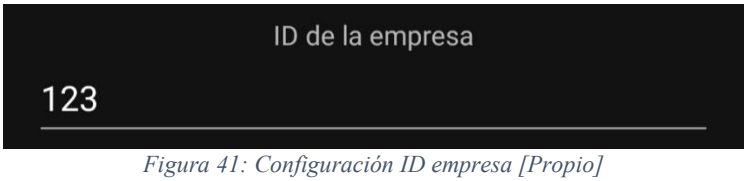

Figura 41: Configuración ID empresa [Propio]

Junto a este campo, se muestra de forma informativa el ID de Android del propio dispositivo. Este identificador único permite al sistema identificar individualmente a cada trabajador a través del Gateway. (Figura 42)

Figura 42: Vista del ID de Android del dispositivo [Propio]

La parte inferior de la vista está dedicada a la configuración del calendario y del horario laboral. Mediante un botón, el usuario puede seleccionar un rango de días en los que tiene previsto trabajar. (Figura 43).

Figura 43: Botón de selección [Propio]

Al pulsar el botón nos emerge una ventana con un calendario, ilustrado en la figura 44. Nos marca el día actual, y en él podemos elegir un día de inicio y uno de fin, para el posterior establecimiento del horario laboral. Al realizar la elección, procedemos a darle a guardar.

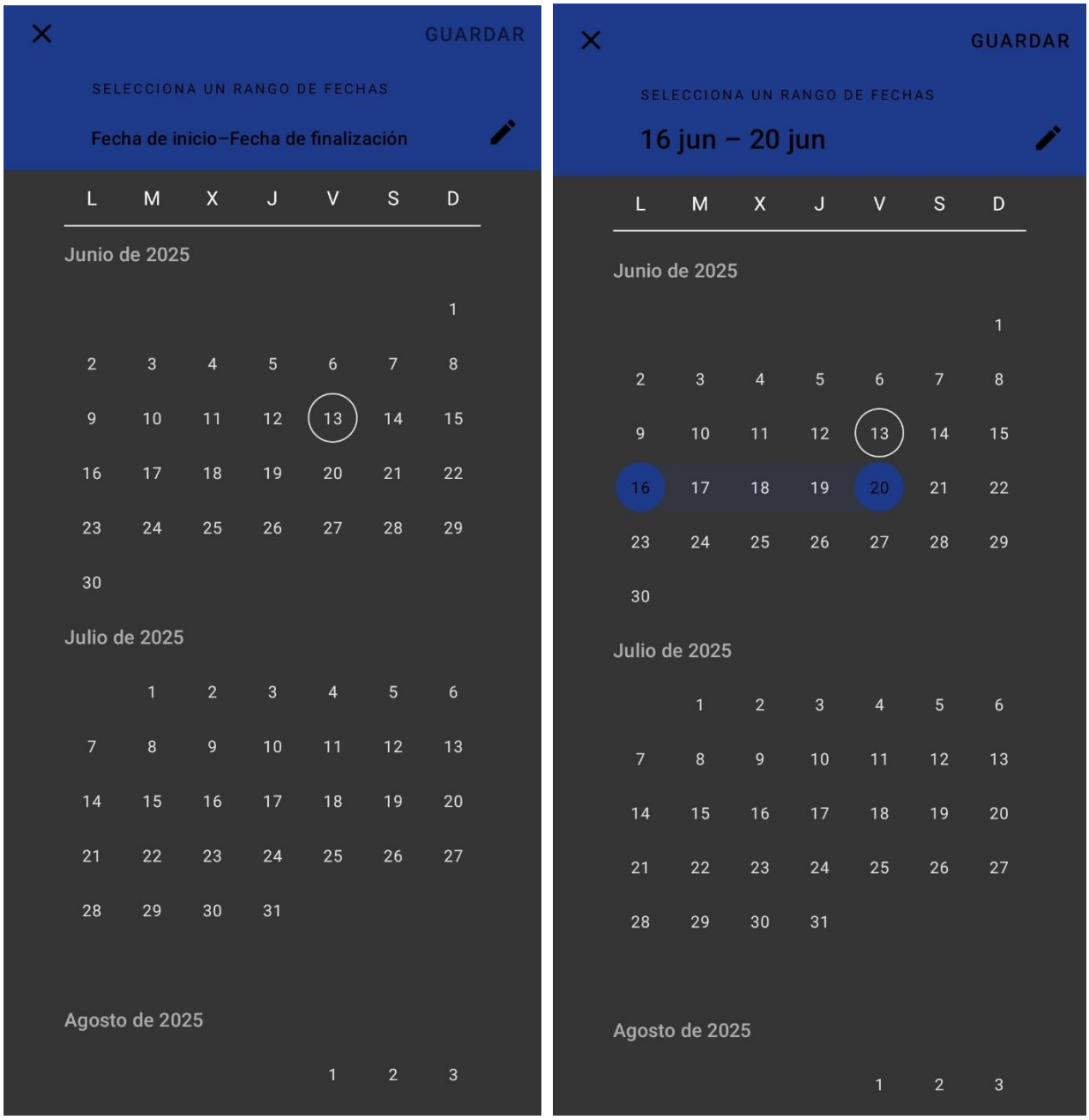

Figura 44: Selección de días [Propio]

Tras pulsar el botón de guardar, emerge la ventana de selección de horario de inicio, como vemos en la figura 45. Primeramente, la ventana mostrará un reloj donde se elige la hora del día. Tras pulsar en ella, la ventana cambiará para seleccionar el minuto de la hora escogida. Para la selección de la hora de fin, pulsamos en aceptar.

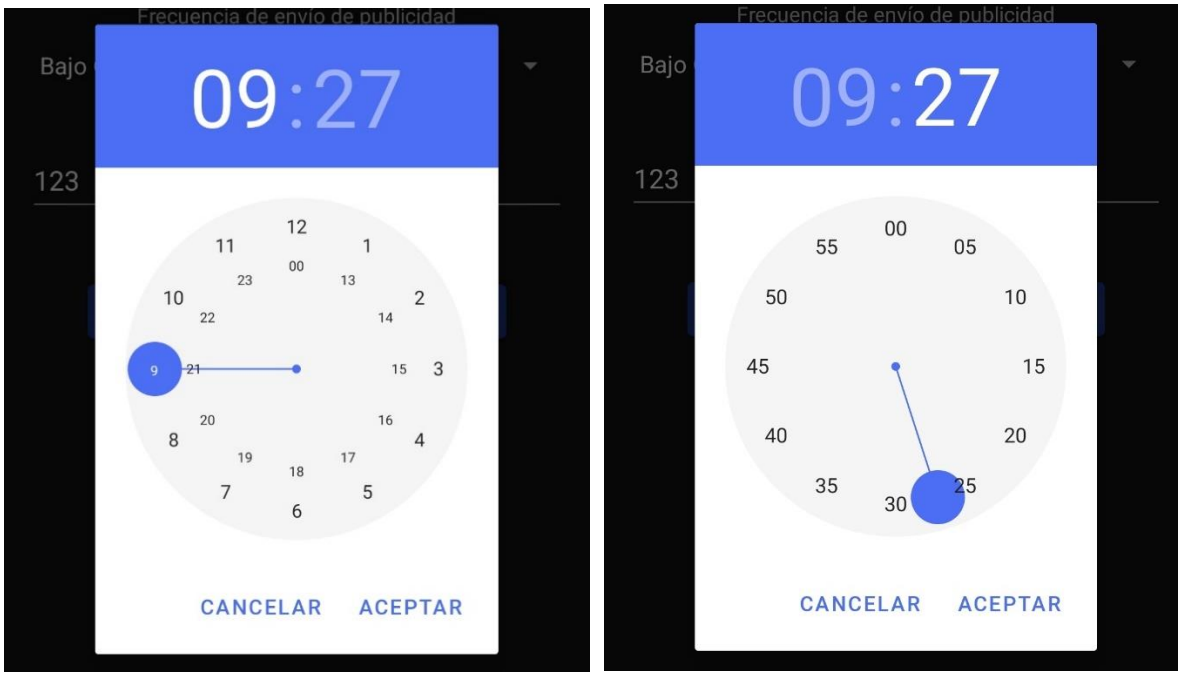

Figura 45: Selección de hora de inicio [Propio]

Como observamos en la figura 46, la ventana emergente es prácticamente igual que la anterior, donde procederemos de la misma manera para la selección de la hora y minuto del fin del horario laboral. Para continuar, pulsamos en aceptar.

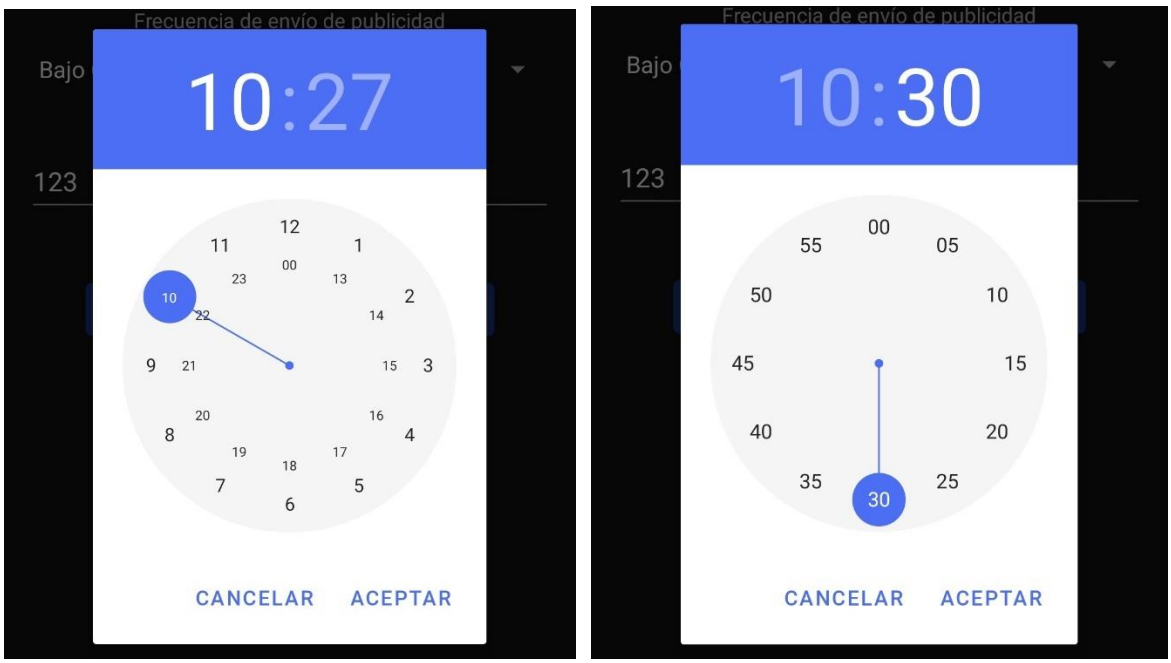

Figura 46: Selección de hora de fin [Propio]

Una vez guardados estos valores, la configuración se muestra en pantalla, listando todos los días y sus respectivos horarios. A medida que los días pasan, los que ya han finalizado desaparecen automáticamente de la vista. Esto podemos verlo en la figura 47.

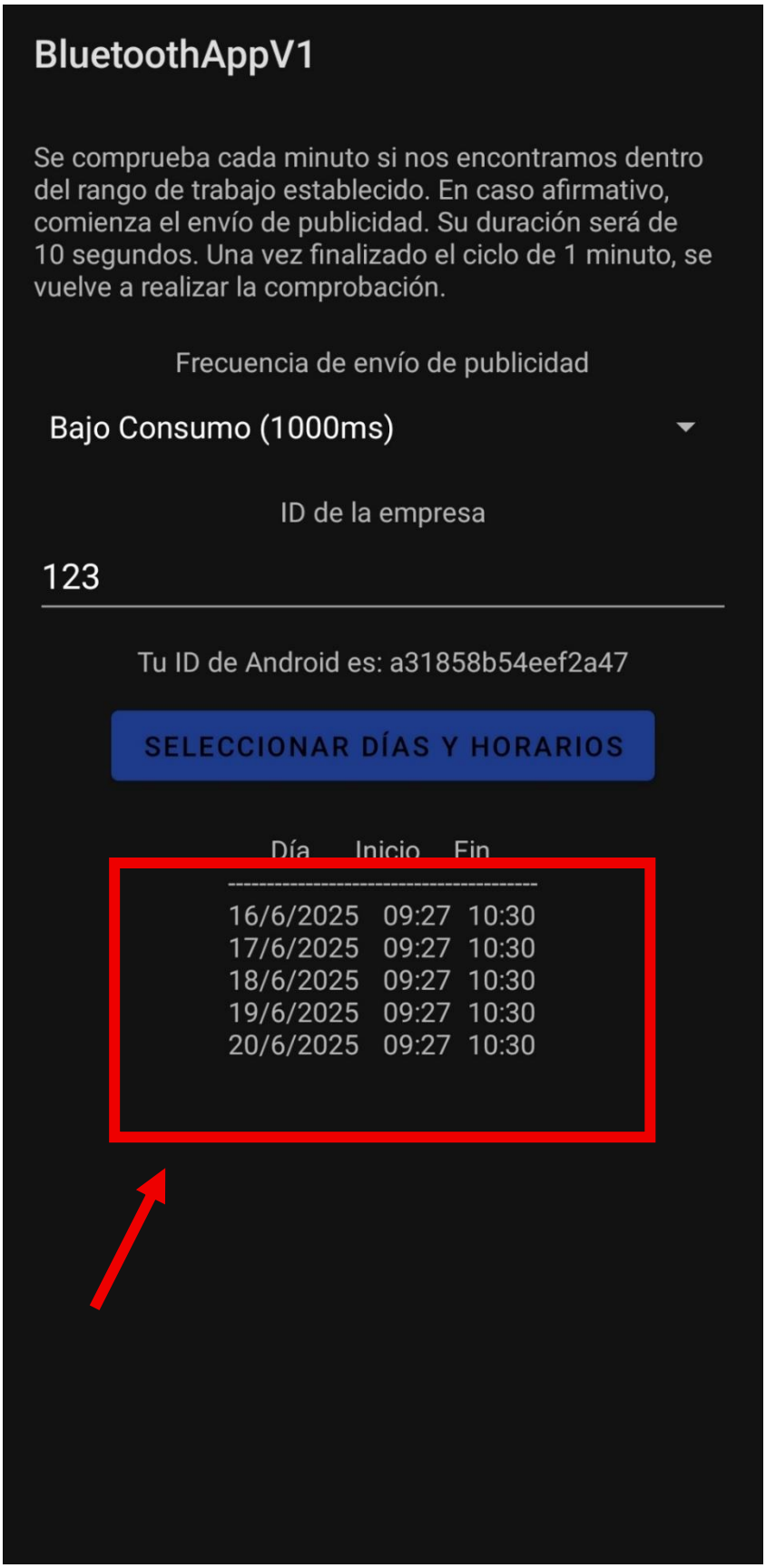

Figura 47: Vista aplicación con calendario establecido [Propio]

La aplicación también incorpora un sistema de notificaciones para informar al usuario sobre su estado de funcionamiento. En primer lugar, se muestra una notificación persistente que indica que la aplicación está corriendo en segundo plano. Esta notificación se mantiene visible para asegurar la continuidad del servicio. En segundo lugar, se genera una notificación específica cuando se está emitiendo publicidad, indicando que se ha entrado en el horario preestablecido y que el Bluetooth está activo. Ambas están mostradas en la figura 48.

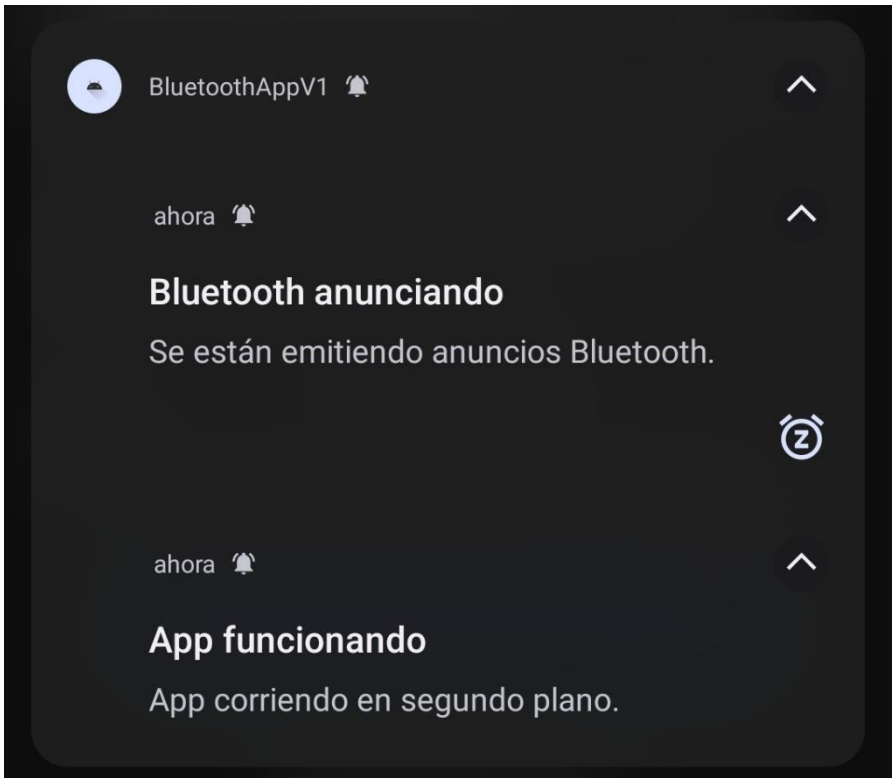

Figura 48: Notificaciones de funcionamiento y emisión [Propio]

Por último, si la aplicación detecta que el trabajador se encuentra en horario de emisión, pero el Bluetooth está desactivado, se mostrará una notificación de advertencia indicando que no ha sido posible iniciar la publicidad por esa causa. (Figura 49)

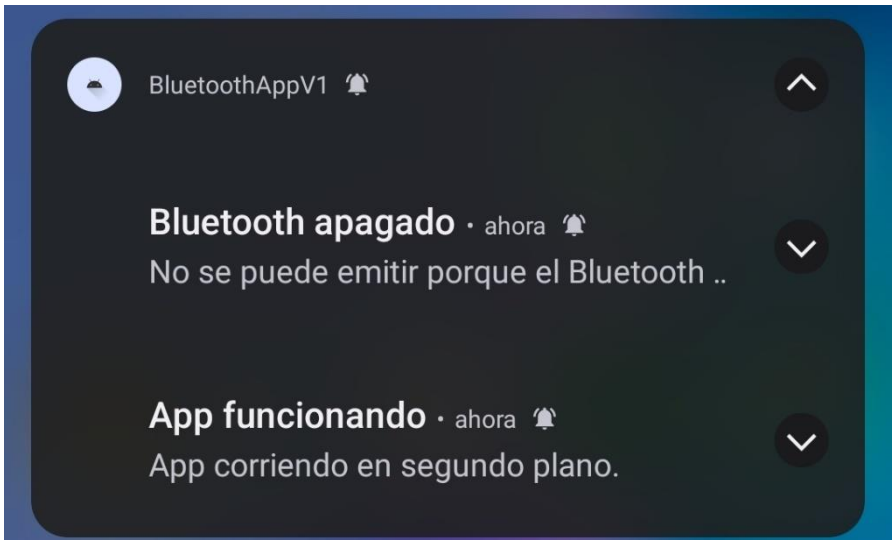

Figura 49: Notificaciones de funcionamiento y bluetooth desactivado [Propio]

El funcionamiento de la aplicación una vez configurada es completamente automático. Cada minuto, comprueba si el momento actual coincide con alguno de los tramos horarios definidos por el trabajador. En caso afirmativo, emite anuncios BLE durante 10 segundos. Cada uno de estos anuncios incluye el nombre del dispositivo, su dirección MAC, el ID de Android y el ID de empresa configurado previamente. La frecuencia de emisión utilizada será la seleccionada por el usuario: 100 ms, 250 ms o 1000 ms. Este proceso se repite de forma continua mientras la aplicación esté en ejecución y las condiciones necesarias estén habilitadas.

ANEXO II: MANUAL DE USUARIO DE APLICACIÓN INTERCONECTADA

Al iniciar la aplicación por primera vez, se muestra una ventana emergente que dirige al usuario a los ajustes de batería del sistema operativo. Esta acción tiene como objetivo excluir la aplicación de las restricciones de optimización energética que podrían detener su funcionamiento en segundo plano, lo cual es fundamental para que la aplicación cumpla su propósito de forma autónoma y continua. Si el usuario configura correctamente esta opción, la advertencia no volverá a aparecer en futuras ejecuciones. (Figura 50)

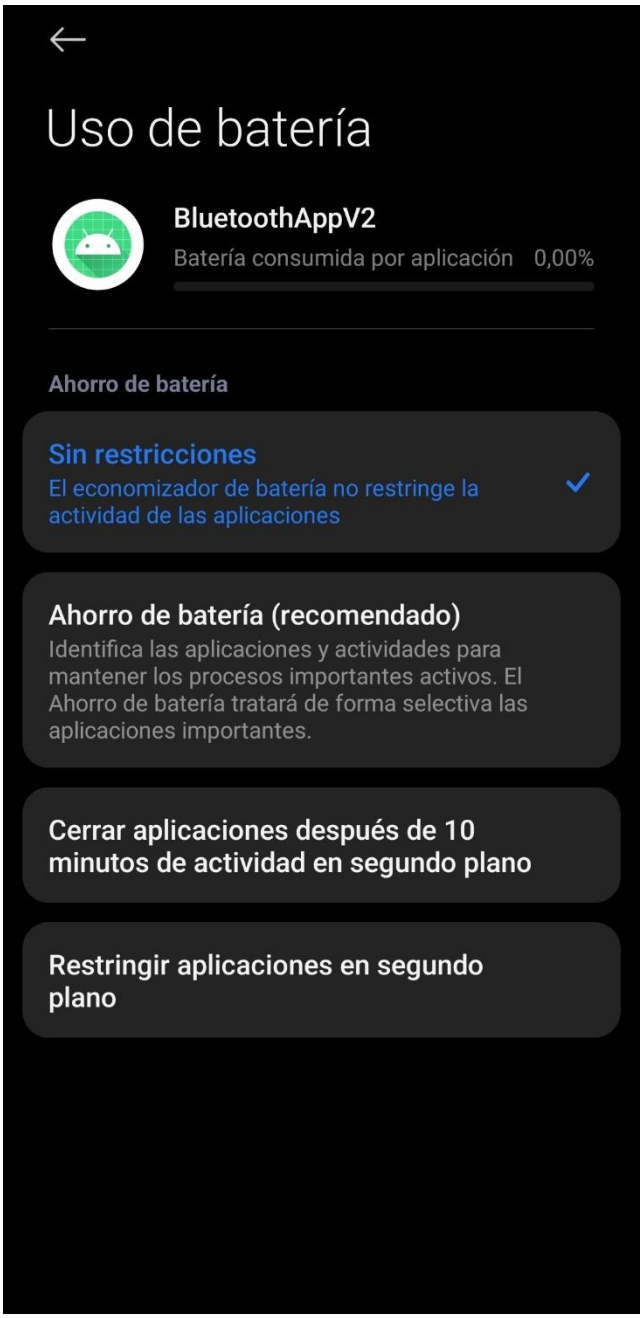

Figura 50: Configuración de batería [Propio]

A continuación, se solicita la concesión de los permisos necesarios para el correcto funcionamiento de la aplicación. En primer lugar, se requiere el permiso de acceso a la ubicación, indispensable para el uso de BLE, ya que Android impone esta condición por motivos de privacidad. Una vez concedido, este permiso no volverá a ser solicitado mientras se mantenga activo. (Figura 51)

Figura 51: Permiso ubicación [Propio]

Seguidamente, se solicita el permiso de acceso a Bluetooth, el cual es absolutamente necesario para permitir cualquier operación de emisión mediante esta tecnología. Sin este permiso, la aplicación no podría realizar su función principal de enviar anuncios BLE. Al aceptar esta solicitud, el permiso se mantendrá de forma persistente y no será solicitado de nuevo mientras no sea revocado. (Figura 52)

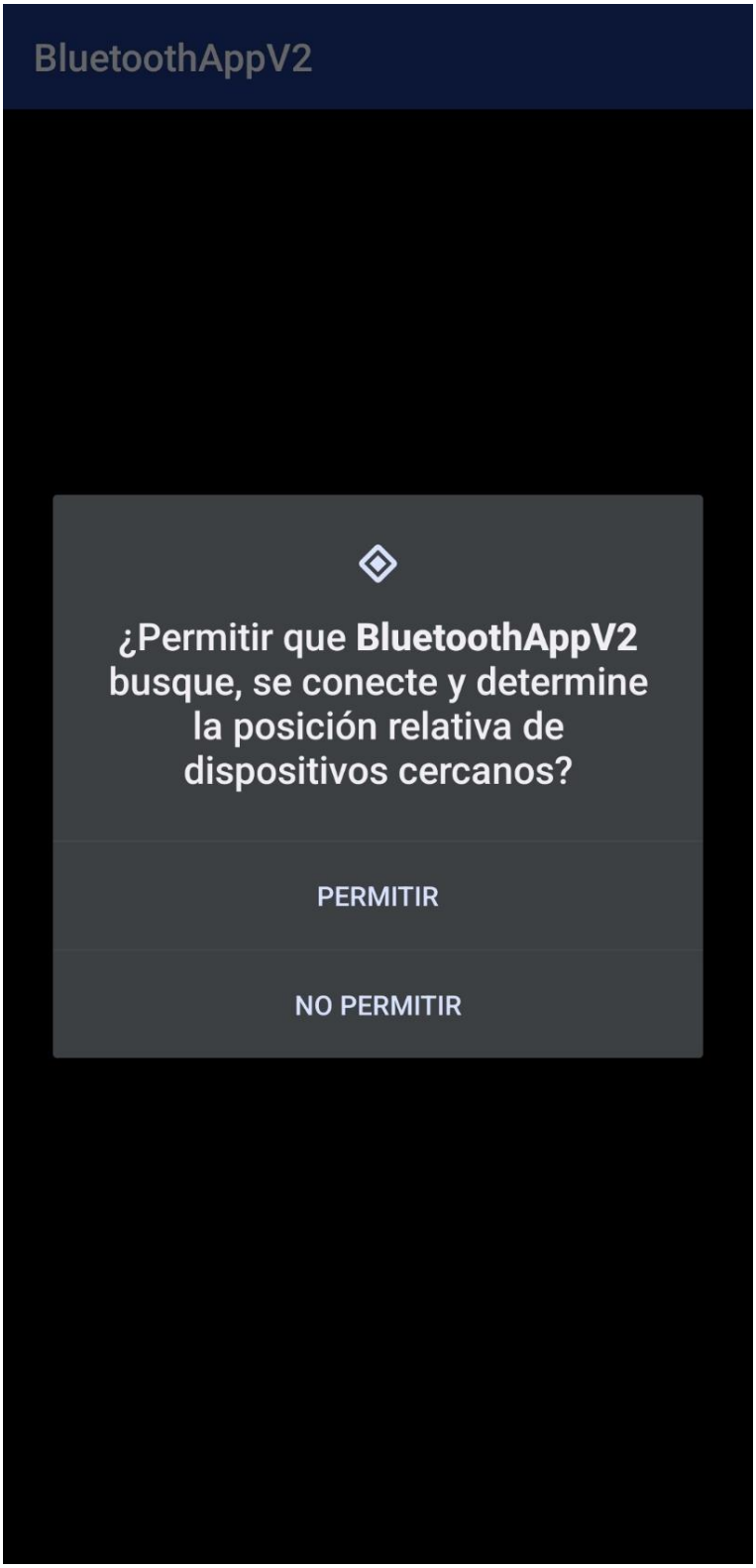

Figura 52: Permiso bluetooth [Propio]

Después de obtener los permisos, la aplicación comprueba si tanto el Bluetooth como la ubicación están activados en el dispositivo. Esta verificación es esencial para su funcionamiento. En caso de que la ubicación esté desactivada, se mostrará una ventana que llevará al usuario directamente a la configuración del sistema para habilitarla manualmente. (Figura 53)

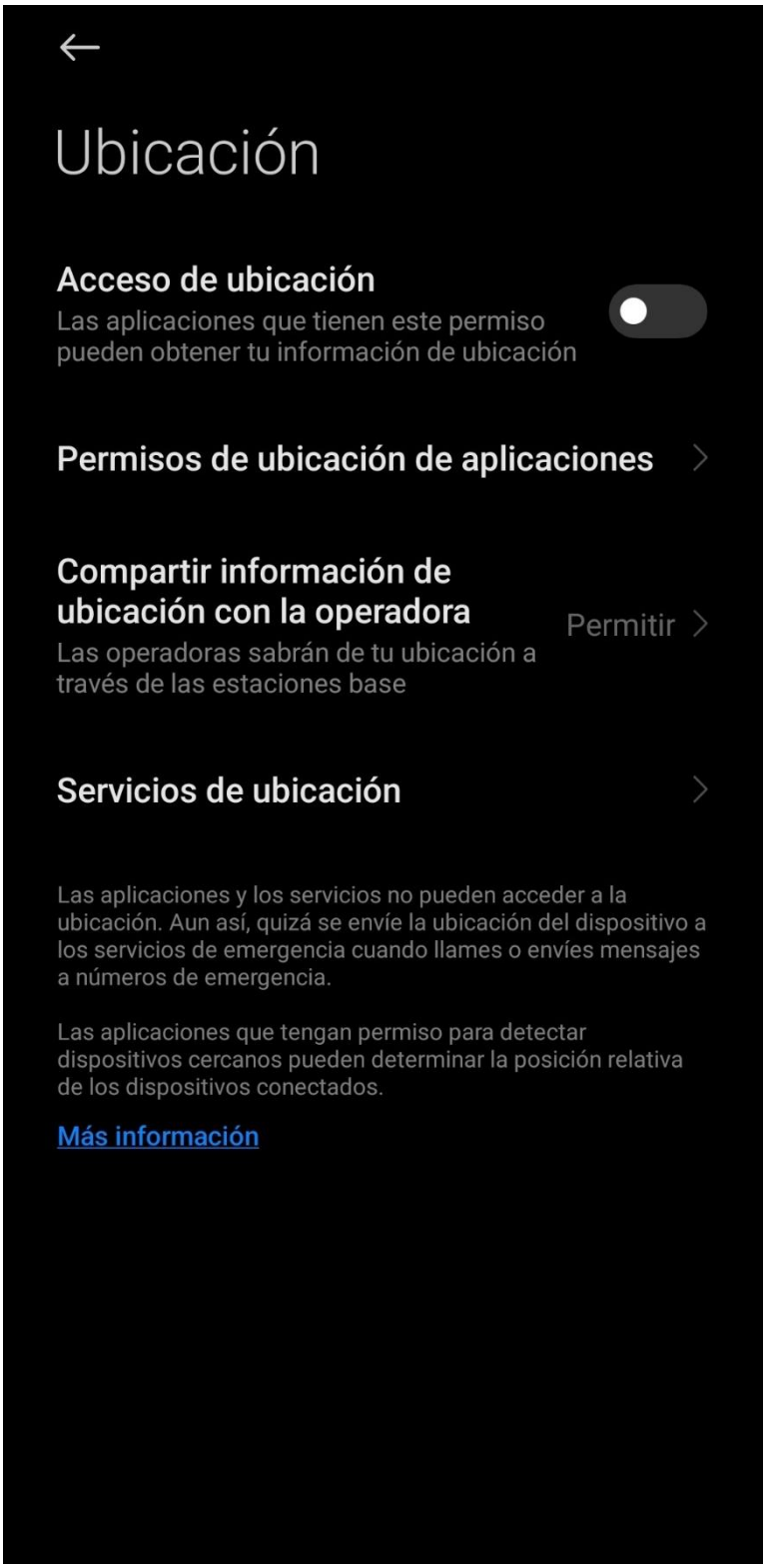

Figura 53: Activar ubicación [Propio]

De manera similar, si el Bluetooth se encuentra desactivado, la aplicación mostrará otra ventana emergente que solicita activarlo. Esta solicitud es necesaria, ya que sin Bluetooth no se podrán emitir los anuncios publicitarios. El usuario deberá pulsar en permitir para habilitar esta función y permitir así que la aplicación comience a operar. (Figura 54)

Figura 54: Activar bluetooth [Propio]

Una vez completados los pasos de concesión de permisos, configuración energética y activación de servicios, se accede a la vista principal de la aplicación. En esta pantalla se muestran varios elementos: el identificador Android del dispositivo, los últimos datos recibidos desde el servidor MQTT y un mensaje que informa al usuario de que la aplicación se encuentra ejecutándose correctamente en segundo plano. (Figura 55)

Figura 55: Vista principal de la aplicación [Propio]

La aplicación cuenta además con un sistema de notificaciones diseñado para mantener al trabajador informado en todo momento sobre el estado de la emisión. En primer lugar, se muestra una notificación persistente que indica que la aplicación está activa y funcionando en segundo plano. Esta notificación no desaparece mientras la aplicación esté en ejecución. En segundo lugar, cuando se encuentra dentro del horario configurado y el Bluetooth está habilitado, se muestra una notificación específica que informa de que se están emitiendo anuncios BLE. Ambas notificaciones son visibles en la figura 56.

Figura 56: Notificaciones de funcionamiento y de emisión [Propio]

En caso de que la aplicación detecte que se ha entrado en horario de emisión, pero que el Bluetooth se encuentra desactivado, se mostrará una tercera notificación de advertencia. Esta informa al usuario de que no ha sido posible emitir publicidad debido a que la conectividad Bluetooth no está habilitada en ese momento. (Figura 57)

Figura 57: Notificaciones de funcionamiento y de bluetooth desactivado [Propio]

El funcionamiento general de la aplicación no requiere ninguna configuración manual ni interacción por parte del trabajador. La aplicación se conecta automáticamente al servidor MQTT, donde permanece a la espera de recibir los parámetros de configuración necesarios: la hora de inicio, la hora de fin de la jornada laboral y el identificador de empresa, que será utilizado posteriormente en el proceso de filtrado por el *Gateway*. En caso de no recibir una nueva configuración, la aplicación continuará funcionando con la última almacenada. El comportamiento del sistema se basa en ciclos de un minuto. En cada ciclo, se verifica si el dispositivo se encuentra dentro del horario establecido. Si es así, se activa la emisión de anuncios BLE durante un periodo de 10 segundos. Este proceso se repite indefinidamente mientras la aplicación permanezca activa y las condiciones requeridas estén cumplidas. En la siguiente figura podemos ver cómo queda la aplicación con la configuración recibida y funcionando. (Figura 58)

Figura 58: Vista de aplicación con datos recibidos desde la conexión MQTT [Propio]

ANEXO III: SCRIPT DE PYTHON PARA ESCANEO AUTÓNOMO

La primera función del *script* tiene como finalidad leer los parámetros de configuración necesarios para ejecutar el proceso de escaneo desde el *Gateway*. Esta configuración se encuentra en el fichero *scan_parameters.txt* e incluye los siguientes elementos: la hora de inicio del escaneo, la hora de finalización, el intervalo de tiempo entre escaneos sucesivos y el identificador de empresa que servirá como filtro para validar los dispositivos detectados. (Figura 59)

```
def read_scan_parameters():
    with open('scan_parameters.txt', 'r') as file:
        lines = file.readlines()
        start_hour = int(lines[0].strip())
        end_hour = int(lines[1].strip())
        time_sleep = int(lines[2].strip())
        FILTER_ID = int(lines[3].strip())

    return start_hour, end_hour, time_sleep, FILTER_ID
```

Figura 59: Función para leer parámetros de fichero local [Propio]

Los datos son extraídos directamente del fichero mencionado, cuyo formato se muestra en la figura 60. En él, la primera línea indica la hora a la que debe comenzar el escaneo; la segunda línea, la hora en la que debe finalizar; la tercera, el tiempo en segundos que debe transcurrir entre escaneos consecutivos; y la cuarta línea, el ID de empresa utilizado para filtrar los anuncios BLE y detectar únicamente a los trabajadores autorizados.

```
10
23
7
123
```

Figura 60: Contenido del fichero local de configuración [Propio]

La segunda función del *script* se encarga de realizar el escaneo BLE del entorno durante un periodo de 20 segundos. Por cada dispositivo detectado, se extraen los datos del fabricante, donde la aplicación móvil incluye tanto el ID de la empresa como el ID único del dispositivo Android. A continuación, se realiza una comparación entre el ID de empresa detectado y el definido en la configuración. Si ambos coinciden, se registra la detección, incluyendo el instante en que se produjo, el nombre del dispositivo, la dirección MAC, la intensidad de la señal (RSSI), los IDs extraídos de los datos del fabricante y los UUIDs de servicios anunciados. Toda esta información se muestra por consola y se almacena en un fichero de texto, lo que permite un seguimiento y análisis posterior de las presencias detectadas. (Figura 61)

```
async def scan_ble(FILTER_ID):
    devices = await BleakScanner.discover(timeout=20.0)
    for device in devices:
        if "manufacturer_data" in device.metadata:
            manufacturer_data = device.metadata["manufacturer_data"]
            for manufacturer_id, data in manufacturer_data.items():
                if FILTER_ID == manufacturer_id:
                    timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
                    result = f"Fecha y hora: {timestamp}\n"
                    result += f"Dispositivo Encontrado: {device.name} - {device.address}\n"
                    result += f"RSSI: {device.rssi} dBm\n"
                    if device.metadata.get("uuids"):
                        result += f"UUIDs de servicio: {device.metadata['uuids']}\n"
                    if "manufacturer_data" in device.metadata:
                        result += f"Manufacturer data: {device.metadata['manufacturer_data']}\n"
                    print(result)
                    print("---------")
                    with open(LOG_FILE, "a") as log_file:
                        log_file.write(result)
```

Figura 61: Función escaneo BLE [Propio]

Finalmente, la última función implementada tiene como propósito verificar si el momento actual se encuentra dentro del horario de escaneo definido en el fichero de configuración. En caso afirmativo, esta función invoca a *scan_ble* para iniciar el proceso de detección. Una vez finalizado el escaneo, el *script* espera el número de segundos establecido en la configuración antes de lanzar una nueva ejecución, manteniendo así un ciclo continuo de comprobación y escaneo mientras se cumplan las condiciones horarias. (Figura 62)

```
def run_for_hours(start_hour, end_hour, time_sleep, FILTER_ID):
    current_time = datetime.now()
    start_time = current_time.replace(hour=start_hour, minute=0, second=0, microsecond=0)
    end_time = current_time.replace(hour=end_hour, minute=0, second=0, microsecond= 0)
    if current_time > end_time:
        end_time += timedelta(days=1)
    print(f"Escaneando desde {current_time.strftime('%H:%M')} hasta {end_time.strftime('%H:%M')}")

    while start_time <=  datetime.now() < end_time:
        asyncio.run(scan_ble(FILTER_ID))
        print("-------Escaneo terminado-------")
        time.sleep(time_sleep)
```

Figura 62: Función para uso continuo y comprobación de hora [Propio]

En la figura 63 se muestra el *script* completo, donde se incluyen las funciones previamente descritas, la importación de los paquetes necesarios, así como la definición y llamada a cada una de las funciones, garantizando así el correcto funcionamiento del sistema.

```
import asyncio
import time
from bleak import BleakScanner
from datetime import datetime, timedelta
LOG_FILE = "scan_results.txt"

def read_scan_parameters():
    with open('scan_parameters.txt', 'r') as file:
        lines = file.readlines()
        start_hour = int(lines[0].strip())
        end_hour = int(lines[1].strip())
        time_sleep = int(lines[2].strip())
        FILTER_ID = int(lines[3].strip())

    return start_hour, end_hour, time_sleep, FILTER_ID


async def scan_ble(FILTER_ID):
    devices = await BleakScanner.discover(timeout=20.0)
    for device in devices:
        if "manufacturer_data" in device.metadata:
            manufacturer_data = device.metadata["manufacturer_data"]
            for manufacturer_id, data in manufacturer_data.items():
                if FILTER_ID == manufacturer_id:
                    timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
                    result = f"Fecha y hora: {timestamp}\n"
                    result += f"Dispositivo Encontrado: {device.name} - {device.address}\n"
                    result += f"RSSI: {device.rssi} dBm\n"
                    if device.metadata.get("uuids"):
                        result += f"UUIDs de servicio: {device.metadata['uuids']}\n"
                    if "manufacturer_data" in device.metadata:
                        result += f"Manufacturer data: {device.metadata['manufacturer_data']}\n"
                    print(result)
                    print("---------")
                    with open(LOG_FILE, "a") as log_file:
                        log_file.write(result)


def run_for_hours(start_hour, end_hour, time_sleep, FILTER_ID):
    current_time = datetime.now()
    start_time = current_time.replace(hour=start_hour, minute=0, second=0, microsecond=0)
    end_time = current_time.replace(hour=end_hour, minute=0, second=0, microsecond= 0)
    if current_time > end_time:
        end_time += timedelta(days=1)
    print(f"Escaneando desde {current_time.strftime('%H:%M')} hasta {end_time.strftime('%H:%M')}")

    while start_time <=  datetime.now() < end_time:
        asyncio.run(scan_ble(FILTER_ID))
        print("-------Escaneo terminado-------")
        time.sleep(time_sleep)

start_hour, end_hour, time_sleep, FILTER_ID = read_scan_parameters()
run for hours(start hour, end hour, time sleep, FILTER ID)
```

Figura 63: Script completo de escaneo autónomo [Propio]

ANEXO IV: SCRIPT DE PYTHON PARA ESCANEO INTERCONECTADO

Antes de detallar las funciones principales del script, es importante describir la conexión del Gateway con el servidor MQTT. Como se puede observar en la figura 64, el primer paso consiste en declarar el cliente MQTT. A continuación, se establece la función de on_message, encargada de gestionar todos los mensajes entrantes desde el servidor. Posteriormente, se realiza la conexión con el servidor MQTT, especificando su dirección IP y el puerto correspondiente. Finalmente, el cliente se suscribe al topic asignado, a través del cual recibirá las configuraciones necesarias para ejecutar el escaneo.

```
client = mqtt.Client()
client.on_message = on_message
client.connect(BROKER_IP, 1883)
client.subscribe(TOPIC)
client.loop_start()
```

Figura 64: Declaración cliente MQTT [Propio]

Una vez establecida la conexión, se pasa a describir el funcionamiento de las distintas funciones del script. La primera de ellas se encarga de decodificar los mensajes recibidos a través de MQTT y extraer los parámetros relevantes. El formato del mensaje está compuesto por la hora de inicio del escaneo, la hora de fin, el intervalo entre escaneos y el identificador de empresa que será utilizado para el filtrado. Estos datos son almacenados en variables globales para que puedan ser reutilizados por el resto de las funciones del *script*. (Figura 65)

```
def on_message(client, userdata, message):
    global start_hour, end_hour, time_sleep, FILTER_ID, start_minute, end_minute
    msg = message.payload.decode("utf-8")
    print(f"Parámetros recibidos: {msg}")
    try:
        msg_clean=msg.strip()
        values = msg_clean.split(",")
        if len(values)!= 4:
            print("El mensaje tiene que tener exactamente 4 valores")
            return
        start_time, end_time, time_sleep_str, FILTER_ID_str = values
        print(f"{start_time} -> {end_time}")
        start_hour, start_minute = map(int, start_time.split(":"))
        end_hour, end_minute = map(int, end_time.split(":"))
        time_sleep = int(time_sleep_str)
        FILTER_ID = int(FILTER_ID_str)
        print(f"{start_hour}:::{start_minute} y {end_hour}:::{end_minute}")

    except ValueError as e:
        print(f"Error: Formato de mensaje incorrecto. Detalles: {e}")
```

Figura 65: Función para recibir el mensaje desde el servidor [Propio]

La segunda función tiene como objetivo realizar el escaneo BLE del entorno durante un periodo de 20 segundos. Por cada dispositivo detectado, se accede a los datos del campo de fabricante, donde la aplicación móvil ha incluido tanto el ID de empresa como el identificador único del dispositivo Android. A partir de esta información, se ejecuta un proceso de comparación entre el ID recibido y el ID de empresa previamente configurado. En caso de coincidencia, se considera una detección válida y se registran diversos datos: el instante en que se produjo la detección, el nombre del dispositivo, la dirección MAC, la intensidad de la señal (RSSI), los identificadores extraídos del campo de fabricante y los UUIDs de los servicios anunciados. Esta información se muestra por consola y se almacena simultáneamente en un fichero de texto, lo que permite realizar un seguimiento cronológico de las presencias detectadas en cada domicilio. (Figura 66)

```
async def scan_ble():
    devices = await BleakScanner.discover(timeout=20.0)
    for device in devices:
        if "manufacturer_data" in device.metadata:
            manufacturer_data = device.metadata["manufacturer_data"]
            for manufacturer_id, data in manufacturer_data.items():
                if FILTER_ID == manufacturer_id:
                    timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
                    result = f"Fecha y hora: {timestamp}\n"
                    result += f"Dispositivo Encontrado: {device.name} - {device.address}\n"
                    result += f"RSSI: {device.rssi} dBm\n"
                    if device.metadata.get("uuids"):
                        result += f"UUIDs de servicio: {device.metadata['uuids']}\n"
                    if "manufacturer_data" in device.metadata:
                        result += f"Manufacturer data: {device.metadata['manufacturer_data']}\n"
                    print(result)
                    print("---------")
                    with open(LOG_FILE, "a") as log_file:
                        log_file.write(result)
```

Figura 66: Función de escaneo BLE [Propio]

La última función implementada en el *script* tiene como misión verificar si se ha recibido una configuración válida desde el servidor y, en ese caso, evaluar si el instante actual se encuentra dentro del intervalo horario de escaneo. Si ambas condiciones se cumplen, la función invoca a *scan_ble* para realizar un nuevo ciclo de detección. Al finalizar, el *script* entra en un estado de espera durante el número de segundos especificado en la configuración, tras lo cual vuelve a ejecutar el proceso, estableciendo así un ciclo automático de comprobación y escaneo, siempre condicionado al horario recibido. (Figura 67)

```python
def run_for_hours():
    global start_hour, end_hour, time_sleep, FILTER_ID, start_minute, end_minute

    while start_hour is None or end_hour is None or start_minute is None or end_minute is None:
        print("Esperando parámetros")
        print(f"{start_hour}:::{start_minute}___{end_hour}:::{end_minute}___{time_sleep}___{FILTER_ID}")
        time.sleep(5)

    current_time = datetime.now()
    end_time = current_time.replace(hour=end_hour, minute=end_minute, second=0, microsecond= 0)
    start_time = current_time.replace(hour=start_hour, minute=start_minute, second=0, microsecond=0)
    if current_time > end_time:
        end_time += timedelta(days=1)

    try:
        while True:
            current_time = datetime.now()
            end_time = current_time.replace(hour=end_hour, minute=end_minute, second=0, microsecond=0)
            start_time = current_time.replace(hour=start_hour, minute=start_minute, second=0, microsecond=0)
            if datetime.now() < end_time and datetime.now() >= start_time:
                print(f"Escaneando desde {current_time.strftime('%H:%M')} hasta {end_time.strftime('%H:%M')}")
                asyncio.run(scan_ble())
                time.sleep(time_sleep)
            else:
                print("Fuera de horario {current_time.strftime('%H:%M')}")
                time.sleep(250.0)

    except KeyboardInterrupt:
        print("Programa detenido")
```

Figura 67: Función para uso continuo y comprobación de hora [Propio]

En las figuras 68 y 69 se muestra el *script* completo, incluyendo las funciones previamente descritas, la importación de los paquetes necesarios, y la lógica de ejecución que permite el correcto funcionamiento del sistema de escaneo BLE sincronizado mediante MQTT.

```python
import asyncio
import time
import paho.mqtt.client as mqtt
from bleak import BleakScanner
from datetime import datetime, timedelta

LOG_FILE = "scan_results.txt"

BROKER_IP = "192.168.1.43"
TOPIC = "bt_scan"

start_hour = None
end_hour = None
time_sleep = None
FILTER_ID = None
end_minute = None
start_minute = None

def on_message(client, userdata, message):
    global start_hour, end_hour, time_sleep, FILTER_ID, start_minute, end_minute
    msg = message.payload.decode("utf-8")
    print(f"Parámetros recibidos: {msg}")
    try:
        msg_clean=msg.strip()
        values = msg_clean.split(",")
        if len(values)!= 4:
            print("El mensaje tiene que tener exactamente 4 valores")
            return
        start_time, end_time, time_sleep_str, FILTER_ID_str = values
        print(f"{start_time} -> {end_time}")
        start_hour, start_minute = map(int, start_time.split(":"))
        end_hour, end_minute = map(int, end_time.split(":"))
        time_sleep = int(time_sleep_str)
        FILTER_ID = int(FILTER_ID_str)
        print(f"{start_hour}:::{start_minute} y {end_hour}:::{end_minute}")

    except ValueError as e:
        print(f"Error: Formato de mensaje incorrecto. Detalles: {e}")

client = mqtt.Client()
client.on_message = on_message
client.connect(BROKER_IP, 1883)
client.subscribe(TOPIC)
client.loop_start()
```

Figura 68: Script completo de escaneo interconectado. Parte 1 [Propio]

```
async def scan_ble():
    devices = await BleakScanner.discover(timeout=20.0)
    for device in devices:
        if "manufacturer_data" in device.metadata:
            manufacturer_data = device.metadata["manufacturer_data"]
            for manufacturer_id, data in manufacturer_data.items():
                if FILTER_ID == manufacturer_id:
                    timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
                    result = f"Fecha y hora: {timestamp}\n"
                    result += f"Dispositivo Encontrado: {device.name} - {device.address}\n"
                    result += f"RSSI: {device.rssi} dBm\n"
                    if device.metadata.get("uuids"):
                        result += f"UUIDs de servicio: {device.metadata['uuids']}\n"
                    if "manufacturer_data" in device.metadata:
                        result += f"Manufacturer data: {device.metadata['manufacturer_data']}\n"
                    print(result)
                    print("---------")
                    with open(LOG_FILE, "a") as log_file:
                        log_file.write(result)


def run_for_hours():
    global start_hour, end_hour, time_sleep, FILTER_ID, start_minute, end_minute

    while start_hour is None or end_hour is None or start_minute is None or end_minute is None:
        print("Esperando parámetros")
        print(f"{start_hour}:::{start_minute}___{end_hour}:::{end_minute}___{time_sleep}___{FILTER_ID}")
        time.sleep(5)

    current_time = datetime.now()
    end_time = current_time.replace(hour=end_hour, minute=end_minute, second=0, microsecond= 0)
    start_time = current_time.replace(hour=start_hour, minute=start_minute, second=0, microsecond=0)
    if current_time > end_time:
        end_time += timedelta(days=1)

    try:
        while True:
            current_time = datetime.now()
            end_time = current_time.replace(hour=end_hour, minute=end_minute, second=0, microsecond=0)
            start_time = current_time.replace(hour=start_hour, minute=start_minute, second=0, microsecond=0)
            if datetime.now() < end_time and datetime.now() >= start_time:
                print(f"Escaneando desde {current_time.strftime('%H:%M')} hasta {end_time.strftime('%H:%M')}")
                asyncio.run(scan_ble())
                time.sleep(time_sleep)
            else:
                print("Fuera de horario {current_time.strftime('%H:%M')}")
                time.sleep(250.0)

    except KeyboardInterrupt:
        print("Programa detenido")

run for hours()
```

Figura 69: Script completo para escaneo interconectado. Parte 2 [Propio]

ANEXO V: SCRIPT DE PYTHON PARA ESCANEO DE BALIZA

La primera función del script tiene como objetivo leer los parámetros de configuración necesarios para la ejecución del proceso de escaneo en el *Gateway*. Estos parámetros se encuentran almacenados en el fichero *scan_parameters2.txt*, el cual incluye: la hora de inicio del escaneo, la hora de finalización, el intervalo de tiempo entre escaneos sucesivos y el identificador de empresa utilizado para filtrar los dispositivos detectados. Estos datos permiten que el sistema funcione únicamente dentro del intervalo previsto y reduzca el número de registros irrelevantes. (Figura 70)

```
def read_scan_parameters():
    with open('scan_parameters2.txt', 'r') as file:
        lines = file.readlines()
        start_hour = int(lines[0].strip())
        end_hour = int(lines[1].strip())
        time_sleep = int(lines[2].strip())
        filter_uuid = lines[3].strip().lower()
    return start_hour, end_hour, time_sleep, filter_uuid
```

Figura 70: Función lectura de parámetros [propio]

Los datos se extraen directamente del fichero mencionado anteriormente, cuyo formato se muestra en la figura 71. En él, la primera línea indica la hora a la que debe comenzar el escaneo; la segunda línea, la hora en la que debe finalizar; la tercera, el tiempo en segundos que debe transcurrir entre escaneos consecutivos; y la cuarta línea, el ID de empresa utilizado para filtrar los anuncios BLE y detectar únicamente a los trabajadores autorizados.

```
7
23
10
12345678901234567890
```

Figura 71: Fichero scan_parameters2.txt

La segunda función se encarga de extraer la información relevante contenida en los anuncios BLE recibidos, concretamente en el campo de datos denominado *Service Data*. De este campo se obtienen los caracteres que representan los valores *instance ID* y *namespace ID*, utilizados para personalizar las emisiones BLE en la aplicación móvil. Esta función permite interpretar adecuadamente la estructura del anuncio y distinguir los emisores válidos del resto de dispositivos presentes en el entorno. (Figura 72)

```
def extract_namespace(service_data_hex):
    """Extrae los primeros 12 bytes después del frame type y txPower (4 bytes)"""
    if service_data_hex.startswith("aafe00") and len(service_data_hex) >= 26:
        namespace_hex = service_data_hex[8:28].lower()  # (10 bytes = 20 hex chars)
        instance_hex = service_data_hex[28:40].lower()
        return namespace_hex, instance_hex
    return None, None
```

Figura 72: Función de extracción de datos del anuncio [propio]

La tercera función está destinada a realizar el escaneo BLE del entorno durante un intervalo de 20 segundos. Por cada dispositivo detectado, se accede al campo *Service Data* del anuncio, donde la

aplicación móvil ha codificado el identificador de empresa. A partir de esta información, se ejecuta una comparación entre el ID recibido y el ID de empresa configurado previamente. Si ambos coinciden, se considera una detección válida y se registran diversos datos: la marca temporal del evento, la dirección MAC del dispositivo, la intensidad de la señal y los identificadores extraídos del campo. Todos estos datos se imprimen por consola y se almacenan de forma local en un fichero de texto, lo que permite llevar un registro ordenado y cronológico de las presencias detectadas en cada vivienda. (Figura 73)

```
def scan_ble(filter_uuid):
    scanner = Scanner()
    devices = scanner.scan(20.0)  # 20 segundos por escaneo

    for dev in devices:
        #found_uuid = False
        service_data = None
        for (adtype, desc, value) in dev.getScanData():
            #if desc == "16b Service UUIDs" and filter_uuid in value.lower():
                #found_uuid = True
            if desc == "16b Service Data":
                service_data = value

        if service_data:
            namespace_id, instance_id = extract_namespace(service_data)
            if namespace_id == filter_uuid:
                timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
                result = f"Fecha y hora: {timestamp}\n"
                result += f"Dispositivo Encontrado: {dev.addr}\n"
                result += f"RSSI: {dev.rssi} dBm\n"
                result += f"Service Data: {service_data}\n"
                result += f"Namespace ID: {namespace_id}\n"
                result += f"Instance ID: {instance_id}\n"
                result += "-----------------------------\n"
                print(result)
                with open(LOG_FILE, "a") as log_file:
                    log_file.write(result)
```

Figura 73: Función de escaneo de la baliza [propio]

Finalmente, la cuarta función tiene como finalidad verificar si el instante actual se encuentra dentro del horario de escaneo definido en el fichero de configuración. Si se cumple esta condición, la función invoca al procedimiento *scan_ble* para iniciar el proceso de detección. Una vez finalizado el escaneo, el *script* entra en estado de espera durante el número de segundos configurado, tras lo cual vuelve a ejecutar una nueva comprobación. De este modo, se establece un ciclo continuo de escaneo periódico que se mantiene activo únicamente dentro de los márgenes horarios establecidos. (Figura 74)

```
def run_for_hours(start_hour, end_hour, time_sleep, filter_uuid):
    current_time = datetime.now()
    start_time = current_time.replace(hour=start_hour, minute=0, second=0, microsecond=0)
    end_time = current_time.replace(hour=end_hour, minute=0, second=0, microsecond=0)
    if current_time > end_time:
        end_time += timedelta(days=1)
    print(f"Escaneando desde {current_time.strftime('%H:%M')} hasta {end_time.strftime('%H:%M')}")

    while start_time <= datetime.now() < end_time:
        scan_ble(filter_uuid)
        print("-------Escaneo terminado--------")
        time.sleep(time_sleep) # tiempo entre escaneos
```

Figura 74: Función para uso continuo y comprobación de hora [Propio]

En la figura 75 se muestra el *script* completo, en el que se incluyen todas las funciones anteriormente descritas, así como la importación de los módulos necesarios y la estructura general del programa. Este conjunto garantiza el funcionamiento autónomo, eficiente y controlado del sistema de escaneo BLE desplegado en el *Gateway*.

```
import time
from datetime import datetime, timedelta
from bluepy.btle import Scanner

LOG_FILE = "scan_results2.txt"

def read_scan_parameters():
    with open('scan_parameters2.txt', 'r') as file:
        lines = file.readlines()
        start_hour = int(lines[0].strip())
        end_hour = int(lines[1].strip())
        time_sleep = int(lines[2].strip())
        filter_uuid = lines[3].strip().lower()
    return start_hour, end_hour, time_sleep, filter_uuid

def extract_namespace(service_data_hex):
    """Extrae los primeros 12 bytes después del frame type y txPower (4 bytes)"""
    if service_data_hex.startswith("aafe00") and len(service_data_hex) >= 26:
        namespace_hex = service_data_hex[8:28].lower()  # (10 bytes = 20 hex chars)
        instance_hex = service_data_hex[28:40].lower()
        return namespace_hex, instance_hex
    return None, None

def scan_ble(filter_uuid):
    scanner = Scanner()
    devices = scanner.scan(20.0)  # 20 segundos por escaneo

    for dev in devices:
        #found_uuid = False
        service_data = None
        for (adtype, desc, value) in dev.getScanData():
            #if desc == "16b Service UUIDs" and filter_uuid in value.lower():
                #found_uuid = True
            if desc == "16b Service Data":
                service_data = value

        if service_data:
            namespace_id, instance_id = extract_namespace(service_data)
            if namespace_id == filter_uuid:
                timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
                result = f"Fecha y hora: {timestamp}\n"
                result += f"Dispositivo Encontrado: {dev.addr}\n"
                result += f"RSSI: {dev.rssi} dBm\n"
                result += f"Service Data: {service_data}\n"
                result += f"Namespace ID: {namespace_id}\n"
                result += f"Instance ID: {instance_id}\n"
                result += "-----------------------------\n"
                print(result)
                with open(LOG_FILE, "a") as log_file:
                    log_file.write(result)
```

```python
def run_for_hours(start_hour, end_hour, time_sleep, filter_uuid):
    current_time = datetime.now()
    start_time = current_time.replace(hour=start_hour, minute=0, second=0, microsecond=0)
    end_time = current_time.replace(hour=end_hour, minute=0, second=0, microsecond=0)
    if current_time > end_time:
        end_time += timedelta(days=1)
    print(f"Escaneando desde {current_time.strftime('%H:%M')} hasta {end_time.strftime('%H:%M')}")

    while start_time <= datetime.now() < end_time:
        scan_ble(filter_uuid)
        print("-------Escaneo terminado-------")
        time.sleep(time_sleep) # tiempo entre escaneos

if __name__ == "__main__":
    start_hour, end_hour, time_sleep, filter_uuid = read_scan_parameters()
    run_for_hours(start_hour, end_hour, time_sleep, filter_uuid)
```

Figura 75: Script completo para el escaneo de la baliza [propio]

ANEXO VI: PASOS PARA PODER EJECUTAR EL SCRIPT

En este anexo se detallan todos los pasos necesarios para configurar correctamente el *Gateway*, incluyendo la instalación de los paquetes y dependencias indispensables para garantizar la compatibilidad con los adaptadores Bluetooth utilizados y permitir la ejecución satisfactoria del *script* de escaneo.

Como primer paso, es recomendable actualizar el sistema para asegurarse de contar con las últimas versiones de los paquetes ya instalados. Posteriormente, se procede a instalar los paquetes relacionados con el funcionamiento del subsistema Bluetooth en Linux. Estos incluyen bluez y bluez-tools, herramientas esenciales para gestionar adaptadores Bluetooth y realizar tareas como emparejamientos o escaneos manuales. Asimismo, es necesario instalar pip, el gestor de paquetes de Python, para poder descargar e instalar la biblioteca bleak, utilizada por el script para realizar escaneos BLE.

Los comandos necesarios para realizar estas operaciones son los siguientes:

```
sudo apt update && sudo apt upgrade -y

sudo apt install bluez bluez-tools

sudo apt install python3-pip

pip install bleak
```

Figura 76: Instalación paquetes necesarios [Propio]

Una vez finalizada la instalación de todos los paquetes requeridos, se procede a la configuración del adaptador Bluetooth. En el caso del adaptador Edimax BT-8500, seleccionado como uno de los modelos recomendados, no es necesario realizar ningún paso adicional, ya que es completamente compatible desde versiones muy tempranas del *kernel* de Linux (a partir de 2.6.32). Este modelo funciona en modo *plug-and-play*, por lo que es detectado automáticamente y está listo para su uso.

En cambio, el adaptador TP-Link UB500, también seleccionado por su mayor alcance, presenta compatibilidad limitada a partir de versiones del *kernel* Linux superiores a la 5.14. Si se utiliza una versión inferior, el adaptador puede ser reconocido por el sistema, pero no será capaz de ejecutar escaneos BLE

correctamente. Por esta razón, en caso de emplear este modelo, será necesario actualizar el *kernel* del sistema a una versión compatible, como por ejemplo la 6.12.22.

El procedimiento para actualizar el kernel comienza por habilitar los repositorios backports, lo que permitirá acceder a versiones más recientes del núcleo de Debian. Para ello, se edita el archivo de fuentes de apt. Dentro del archivo, se debe añadir la siguiente línea (en caso de que no esté presente). Se guarda el archivo y se actualizan los repositorios.

```
sudo nano /etc/apt/sources.list

➔ deb http://deb.debian.org/debian bookworm-backports main contrib non-free non-free-firmware

sudo apt update
```

Figura 77: Habilitación repositorios [Propio]

Tras esto, procedemos a instalar el *kernel* 6.12.22. Para ello, buscamos el paquete correspondiente y elegimos el siguiente de entre los resultados. Una vez elegido, lo descargamos en el directorio respectivo.

```
apt-cache apt-cache search linux-image | grep 6.12

➔ linux-image-6.12.22+bpo-amd64 - Linux 6.12 for 64-bit PCs (signed)

cd /lib/modules

sudo apt install -t bookworm-backports linux-image-6.12.22+bpo-amd64
```

Figura 78: Instalación del kernel [Propio]

Después de la instalación, se debe reiniciar el sistema y seleccionar manualmente el nuevo *kernel* desde el menú de GRUB. Durante el arranque, se pulsa una tecla para acceder al menú, se elige la opción "Advanced options for Debian GNU/Linux", y dentro de ella se selecciona el *kernel* recién instalado. Para comprobar que el sistema ha arrancado correctamente con el nuevo núcleo, se puede ejecutar:

```
uname -r
```

Figura 79: Comprobar kernel actual [Propio]

Con el nuevo *kernel* activo, se procede a instalar los controladores del adaptador TP-Link. En caso de ser necesario, se puede crear un directorio dedicado para ello y utilizar *curl* u otra herramienta para descargar los *drivers* desde el repositorio del fabricante o un repositorio comunitario confiable. Una vez descargados, se instalan y se recarga el módulo del sistema correspondiente.

```
sudo mkdir -p /lib/firmware/rtl_bt

sudo apt install curl
```

Figura 80: Preparación para instalar los drivers [Propio]

```
sudo curl -s https://git.kernel.org/pub/scm/linux/kernel/git/firmware/linux-firmware.git/plain/rtl_bt/rtl8761bu_fw.bin -o /lib/firmware/rtl_bt/rtl8761bu_fw.bin

sudo curl -s https://git.kernel.org/pub/scm/linux/kernel/git/firmware/linux-firmware.git/plain/rtl_bt/rtl8761bu_config.bin -o /lib/firmware/rtl_bt/rtl8761bu_config.bin
```

Figura 81: Instalación de los drivers necesarios [Propio]

```
sudo modprobe -r btusb
sudo modprobe btusb
```

Figura 82: Recarga del módulo de bluetooth [Propio]

Para comprobar que el Bluetooth funciona correctamente, primero listamos los adaptadores Bluetooth disponibles en el sistema. Desactivamos los que no nos interesan (si los hay) y probamos a escanear desde la línea de comandos. Una vez comprobado que funciona, ejecutamos el *script* (desactivando el filtro si es necesario) para probar que todo funciona correctamente.

```
sudo hciconfig -a
sudo hciconfig hciX down
bluetoothctl scan on
python3 scan_ble_AppV1.py
```

Figura 83: Comprobación de funcionamiento de escaneo BLE y del script [Propio]