Grado en Ingeniería en Tecnologías de Telecomunicación

Trabajo de Fin de Grado

Sistema colaborativo de detección ampliada de drones

Autor
Ignacio Garrastazu Fernández

Director
Atilano Ramiro Fernández-Pacheco Sánchez-Migallón

Madrid
Junio 2025

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título *Sistema colaborativo de detección ampliada de drones* en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico Grado en Ingeniería en Tecnologías de Telecomunicación es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Ignacio Garrastazu Fernández
Fecha: 29 / 06 / 2025

Autorizada la entrega del proyecto
EL DIRECTOR DEL PROYECTO

71216314B ATILANO RAMIRO FERNÁNDEZ-PACHECO

Digitally signed by 71216314B ATILANO RAMIRO FERNÁNDEZ-PACHECO
Date: 2025.06.30 14:02:58 +02'00'

Fdo.: Atilano Ramiro Fernández-Pacheco Sánchez-Migallón
Fecha: 29 / 06 / 2025

Grado en Ingeniería en Tecnologías de Telecomunicación

Trabajo de Fin de Grado

Sistema colaborativo de detección ampliada de drones

Autor
Ignacio Garrastazu Fernández

Director
Atilano Ramiro Fernández-Pacheco Sánchez-Migallón

Madrid
Junio 2025

Resumen

Este Trabajo de Fin de Grado presenta el desarrollo de un sistema para la detección, visualización y almacenamiento de drones equipados con balizas compatibles con el estándar Open Drone ID. El sistema se compone de una aplicación móvil desarrollada con Flutter, capaz de escanear tramas Bluetooth Low Energy (BLE) emitidas por las balizas de los drones, y de un *backend* implementado en Java con Spring Boot, que expone una API REST y persiste la información en una base de datos relacional (H2). El sistema permite registrar usuarios, iniciar sesión, recibir tramas de los drones para localizarlos y enviar sus datos a la base de datos, a la vez que recibe los datos de los drones ya existentes. A lo largo del proyecto se han aplicado principios de diseño por capas, buenas prácticas de ingeniería del software y técnicas de comunicación cliente-servidor.

Estado de la cuestión

El problema encontrado por la empresa Drone by Drone [1] es que el alcance Wi-Fi/Bluetooth de las balizas no es suficiente al trabajar en un área saturada de drones.

La idea propuesta es el desarrollo de un sistema que sea capaz de almacenar la localización del dron detectado por el dispositivo y de enviar esa localización a todos los dispositivos que tengan acceso al sistema. Este proyecto puede ayudar a que se logre aumentar el alcance de detección de drones, especialmente drones dentro de la misma organización que utilice este sistema con varios dispositivos.

Arquitectura General del Sistema

La arquitectura del sistema sigue el modelo cliente-servidor. Estas partes son las siguientes:

1. **La baliza Dronetag Beacon** instalada en el dron, que emite paquetes BLE con su identificación y telemetría.
2. **La aplicación móvil Flutter (*frontend*)**, que escanea los anuncios BLE, parsea la trama según la especificación Open Drone ID y envía la información al servidor mediante peticiones HTTP REST en formato JSON. También actúa como la capa de interfaz de usuario (cliente)
3. **El *backend* Spring Boot**, organizado en capas (controlador, servicio, repositorio y modelo) y responsable de recibir las peticiones, aplicar la lógica de negocio. Además es el encargado de acceder a la persistencia.
4. **La base de datos relacional H2**, donde los datos se almacenan a través de Spring Data JPA.

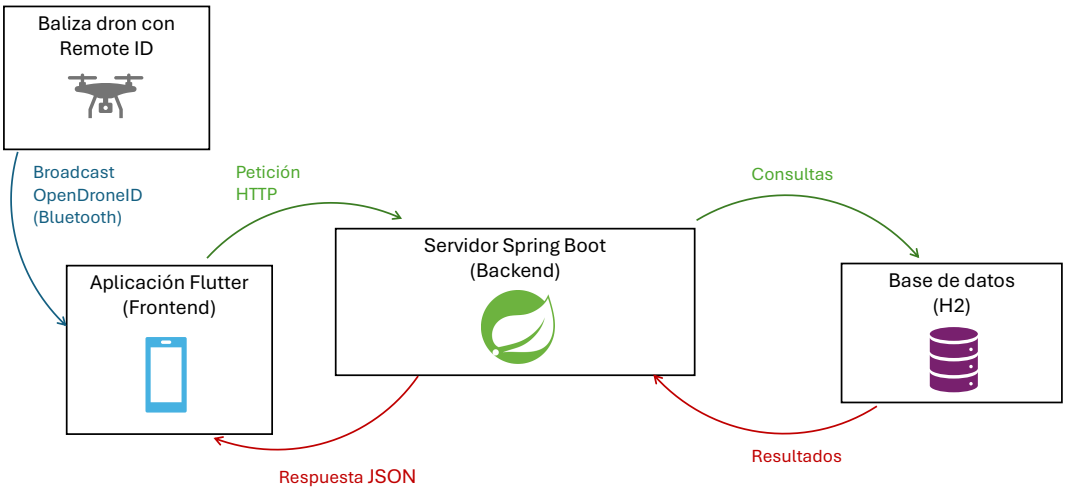

Figura 1: Diagrama de caja negra.

Diagrama de casos de uso

El diagrama de casos de uso [2] muestra la manera en la que el sistema interactúa con entidades externas.

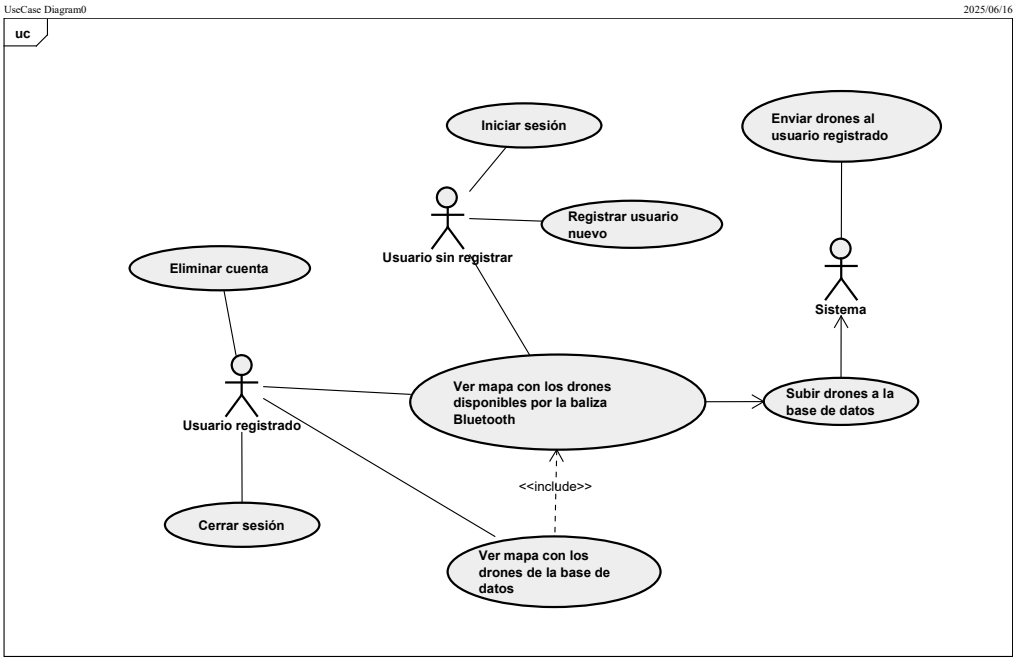

Figura 2: Diagrama de casos de uso

Integración del sistema

La integración del sistema se ha llevado a cabo siguiendo una arquitectura cliente-servidor. A continuación, se detallan los principales aspectos de esta integración:

- **Integración entre la aplicación móvil y el *backend***: La aplicación Flutter se comunica con el servidor *backend* a través de peticiones HTTP a la API REST expuesta por el sistema.
- **Integración de la detección BLE y la interfaz**: La aplicación móvil implementa un servicio de escaneo BLE que detecta en tiempo real las tramas emitidas por balizas compatibles con el estándar Open Drone ID.
- **Interacción con la base de datos**: El *backend* desarrollado en Spring Boot emplea Spring Data JPA para mapear entidades Java a una base de datos relacional H2.

- **Persistencia de información**: A la vez almacenados, los datos sobre drones detectados en otros envíos se incluyen en la respuesta de la petición para garantizar una respuesta rápida con una sola petición.

Conclusión

El desarrollo de este Trabajo de Fin de Grado ha permitido diseñar e implementar un sistema colaborativo de detección de drones equipados con balizas de identificación remota compatibles con el estándar Open Drone ID. Se ha abarcado tanto el desarrollo móvil como el diseño de arquitecturas cliente-servidor, la gestión de comunicaciones Bluetooth Low Energy (BLE) y el modelado de bases de datos relacionales mediante Spring Boot. Se ha logrado implementar una aplicación móvil desarrollada con Flutter, capaz de escanear y decodificar tramas BLE emitidas por las balizas de los drones, mostrando en tiempo real la información recibida. Esta información incluye datos clave como la posición, la hora de recepción del mensaje y el identificador único del dron, todo ello presentado mediante una interfaz gráfica intuitiva. Además, la aplicación permite a los usuarios registrarse, iniciar sesión y envía automáticamente la información detectada con el resto del sistema, fomentando así una red colaborativa de detección de drones.

Abstract

This Final Degree Project presents the development of a system for the detection, visualization, and storage of drones equipped with beacons compatible with the Open Drone ID standard. The system consists of a mobile application developed in Flutter, capable of scanning Bluetooth Low Energy (BLE) frames emitted by the drones' beacons, and a *backend* implemented in Java with Spring Boot, which exposes a REST API and persists the information in a relational database (H2). The system allows user registration, login, receiving frames from drones to locate them, and sending their data to the database, while also receiving data of existing drones. Throughout the project, layered design principles, software engineering best practices, and client-server communication techniques have been applied.

State of the Art

The problem identified by the company Drone by Drone [1] is that the Wi-Fi / Bluetooth range of the beacons was not sufficient when operating in an area saturated with drones.

The proposed idea is the development of a system capable of storing the location of a drone detected by a device and sharing that location with all devices that have access to the system. This project may help to increase the detection range of drones, especially those belonging to the same organization using this system across multiple devices.

General System Architecture

The system architecture follows a client-server model. The main components are as follows:

1. **The Dronetag Beacon**, installed on the drone, which emits BLE packets containing its identification and telemetry data.
2. **The Flutter mobile application (frontend)**, which scans for BLE advertisements, parses the frames according to the Open Drone ID specification, and sends the information to the server using HTTP REST requests in JSON format. It also serves as the user interface layer (client).
3. **The Spring Boot backend**, organized in layers (controller, service, repository, and model), responsible for handling requests, applying business logic, and managing data persistence.
4. **The H2 relational database**, where data is stored through Spring Data JPA.

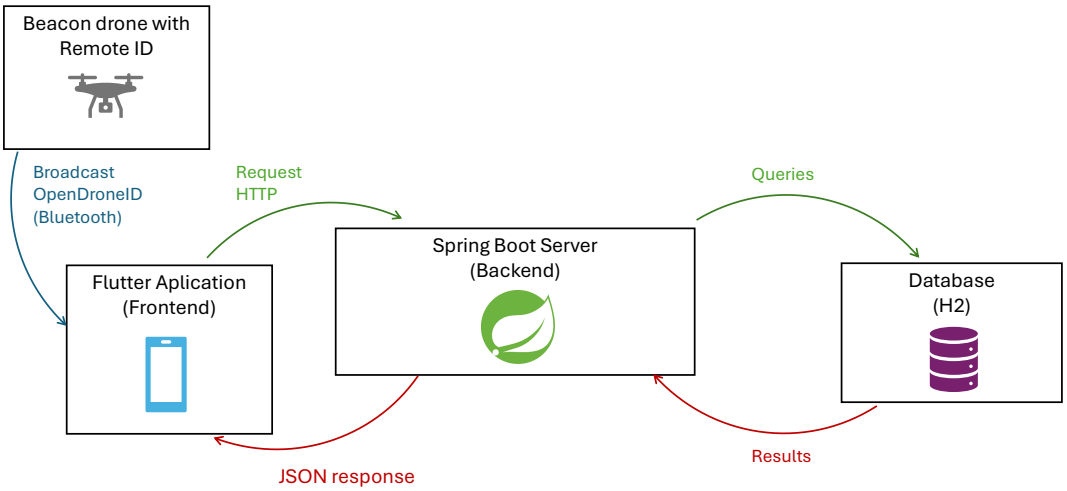

Figure 3: Black-box diagram.

Use Case Diagram

The use case diagram [2] illustrates how the system interacts with external entities.

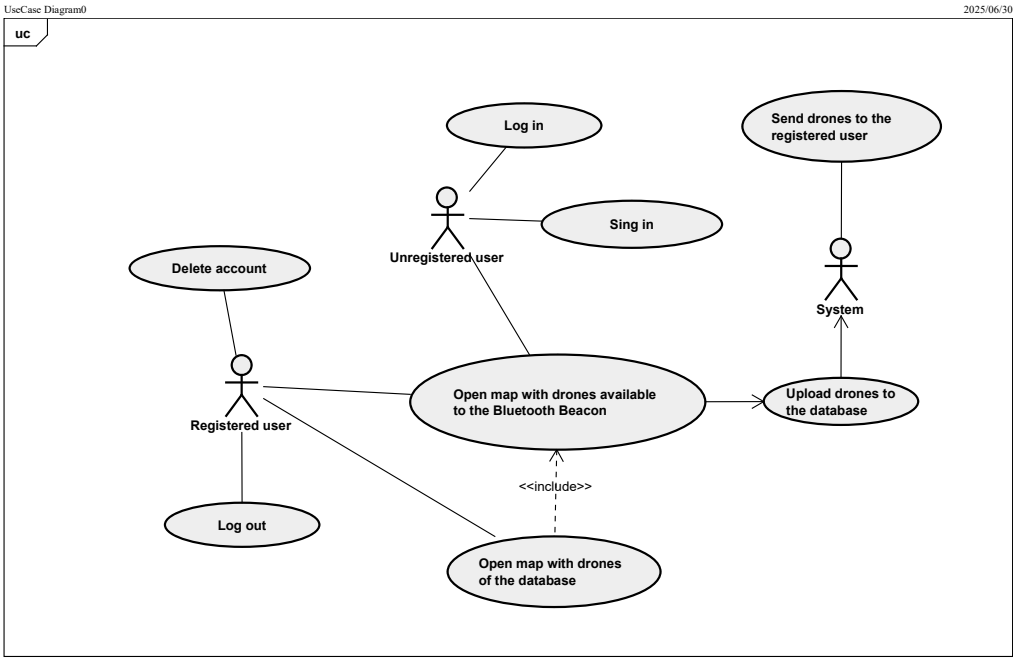

Figure 4: Use case diagram

System Integration

The system integration was carried out following a client-server architecture. The main aspects of this integration are described below:

- **Integration between the mobile application and the backend**: The Flutter application communicates with the backend server through HTTP requests to the REST API exposed by the system.
- **Integration of BLE detection and the user interface**: The mobile application implements a BLE scanning service that detects, in real time, the frames emitted by beacons compliant with the Open Drone ID standard.
- **Interaction with the database**: The *backend* developed with Spring Boot uses Spring Data JPA to map Java entities to a relational H2 database.
- **Data persistence**: Once stored, information about drones detected in other transmissions is included in the response to the request, ensuring a fast response with a single query.

Conclusion

The development of this Final Degree Project has made it possible to design and implement a collaborative drone detection system for drones equipped with remote identification beacons compatible with the Open Drone ID standard. The work encompassed both mobile development and client-server architecture design, management of Bluetooth Low Energy (BLE) communications, and modeling of relational databases using Spring Boot.

The result is a mobile application developed in Flutter, capable of scanning and decoding BLE frames emitted by drone beacons and displaying the received information in real time. This data includes key elements such as the drone's position, the timestamp of message reception, and its unique identifier, all presented through an intuitive graphical interface. Additionally, the application allows users to register, log in, and automatically share detected information with the rest of the system, thus fostering a collaborative drone detection network.

Agradecimientos

Me gustaría agradecer a mi familia por proporcionarme la educación necesaria para la elaboración de este Trabajo de Fin de Grado y por ofrecerme todos los recursos que he necesitado. A la empresa Drone By Drone por ofrecerme su ayuda y el equipamiento necesario. A Antía y a Javier por acompañarme en el proceso y apoyarme con el desarrollo del proyecto cuando mi familia estaba demasiado lejos como para hacerlo.

Índice general

Índice de figuras

Índice de tablas

Índice de Códigos

Capítulo 1

Introducción

Los drones, también conocidos como vehículos aéreos no tripulados (UAV, por sus siglas en inglés) han evolucionado a lo largo de los últimos años debido a su versatilidad en diferentes campos.

Según la Agencia Estatal de Seguridad Aérea (AESA), hasta febrero de 2018, el sector en el que los drones eran más utilizados era para la realización de Fotografía, Filmación y Levantamiento [3]. En la Figura 1.1 se encuentran los sectores en los que los drones son utilizados.

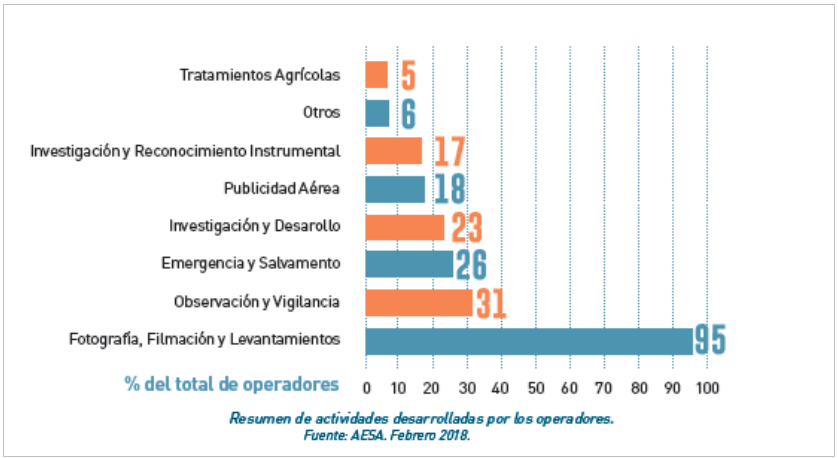

Figura 1.1: Actividades desarrolladas por los operadores [3].

Aun así, este no es el único uso que los drones tienen. Algunos ejemplos de estos nuevos usos son en el ámbito militar, mapeo 3D, mantenimiento de infraestructura, control de desastres naturales, construcción y minería, entre muchos otros [4]. Estos dispositivos revolucionan la forma en que se abordan numerosos desafíos, en especial en el ámbito profesional, y siguen ampliando sus aplicaciones,

posicionándose como aliados clave en la transformación digital de múltiples sectores.

Con el crecimiento de estas tecnologías, ha crecido también la legislación acerca de estas. Un ejemplo de esto es la nueva necesidad de identificar a estos drones. Desde el 1 de enero de 2024, todos los drones deberán tener un sistema actualizado y aprobado por la EASA (European Union Aviation Safety Agency) de identificación remota [5]. Esta identificación se denomina *Remote ID*, y funciona utilizando las tecnologías Wi-Fi y Bluetooth para la transmisión de datos del UAV.

Este sistema de identificación remota ha traído un abanico de posibilidades a la hora de la realización de sistemas que utilicen esta información transmitida en *broadcast*, como el sistema estandarizado por la ASTM [1] Open Drone ID y el uso de aplicaciones móviles para la localización de drones.

[1]La ASTM International es una organización reconocida por establecer normas técnicas voluntarias a nivel global [6]

Capítulo 2

Descripción de las Tecnologías

2.1. Descripción de las tecnologías

Este capítulo presenta los componentes tecnológicos empleados en el proyecto, justificando su elección y resaltando las características que aportan valor a la solución.

2.1.1. Flutter

Flutter es un *SDK* de código abierto de Google que permite compilar aplicaciones nativas para Android, iOS, web y escritorio a partir de un único código fuente en Dart. Su motor gráfico propio (Skia) dibuja la interfaz directamente sobre la GPU. Gracias a esto se logran un rendimiento cercano al nativo y un refresco fluido a 60 fps o más. Incluye un sistema de *hot-reload* muy ágil y un ecosistema amplio de paquetes comunitarios [7].[1]

Figura 2.1: Logotipo de Flutter [7].

El motivo de la elección de este SDK viene con la necesidad de un *frontend* móvil ligero, además de su gran contenido de librerías y su gran curva de aprendizaje.

Dentro de Flutter, fueron utilizadas las siguientes librerías:

[1] La documentación oficial de Flutter se encuentra en `https://flutter.dev`.

- **Google Maps Flutter:** utilizada para todo lo relacionado con el mapa, incluidos los marcadores.
- **Geolocator**: utilizada para obtener la posición del usuario.
- **Permission Handler**: utilizada para la solicitud de los permisos necesarios para el correcto funcionamiento de la aplicación.
- **Http Dart:** utilizada para el intercambio de mensajes con el servidor.
- **Flutter Opendroneid [8]:** necesaria para la interpretación de mensajes de los drones utilizando la tecnología estandarizada Open Drone ID y encargada de gestionar el escaneo Bluetooth (Un ejemplo del uso de esta librería se encuentra en el Código 5.9).
- **Go route:** es una librería oficial de Flutter que facilita la gestión de rutas y la navegación entre pantallas de forma declarativa.
- **Shared preferences:** esta librería proporciona una forma sencilla de almacenar pares clave-valor de forma persistente en el dispositivo del usuario. Se utiliza para guardar el token de sesión. Internamente utiliza el almacenamiento nativo del sistema (como `NSUserDefaults` en iOS o `SharedPreferences` en Android) [9] (En el Código 5.7 se puede ver un ejemplo de esta librería).
- Otras librerías integradas en Flutter.

```
dependencies:
  flutter:
    sdk: flutter
  http: ^1.3.0
  google_maps_flutter: ^2.12.1
  geolocator: ^13.0.4
  json_serializable: ^6.9.4
  flutter_blue_plus: ^1.26.4
  permission_handler: ^11.2.0
  flutter_opendroneid: ^0.22.1
  go_router: ^13.2.2
  shared_preferences: ^2.2.2
```

Código 2.1: Dependencias de Flutter con las versiones correspondientes (del archivo `pubspec.yaml`).

2.1.2. Dart

Dart es un lenguaje tipado y orientado a objetos que compila tanto a código nativo como a JavaScript. Destaca su modelo asíncrono basado en `Future/Stream` y el uso de un *garbage collector* optimizado para interfaces reactivas [10]. Al ser el lenguaje oficial de Flutter, su curva de adopción es muy buenq y asegura compatibilidad total con el SDK.

2.1.3. Bluetooth Low Energy y OpenDroneID

La baliza instalada en el dron emite tramas *Bluetooth Low Energy* (BLE) en formato Open Drone ID, un estándar impulsado por la ASTM y la FAA [2] para la identificación remota de aeronaves no tripuladas.

Este formato de identificación remota (Open Drone ID), fue definido por ASTM F3411 [12]. Este estándar es el utilizado también por fabricantes como Dronetag [13], cuyos módulos comerciales (como Dronetag Mini o Beacon) emiten las tramas BLE en el mismo formato. Es por esto que se ha optado por este protocolo. (En la Figura 2.2 se pueden ver la trama en BLE de Open Drone ID).

Los mensajes de Open Drone ID se transmiten en diferentes formatos, dependiendo del tipo de información que contienen. Los principales tipos de mensajes incluyen:

- **Basic ID:** proporciona una identificación básica del dron, como su número de serie o identificador de operador.
- **Location**: incluye información sobre la ubicación actual del dron, como latitud, longitud, altitud, velocidad y dirección.
- **Authentication**: contiene datos para autenticar la identidad del dron.
- **Self-ID**: permite al operador incluir información adicional, como el propósito del vuelo.
- **System**: proporciona detalles sobre el sistema del dron, incluyendo información sobre el operador y la categoría del dron.

En el cliente Flutter se emplea el paquete `flutter_opendroneid` para escanear, decodificar y validar las tramas [8].

[2] La FAA (Federal Aviation Administration) es la Administración Federal de Aviación de los Estados Unidos, una agencia del Departamento de Transporte (DoT) del gobierno federal de EE. UU. [11].

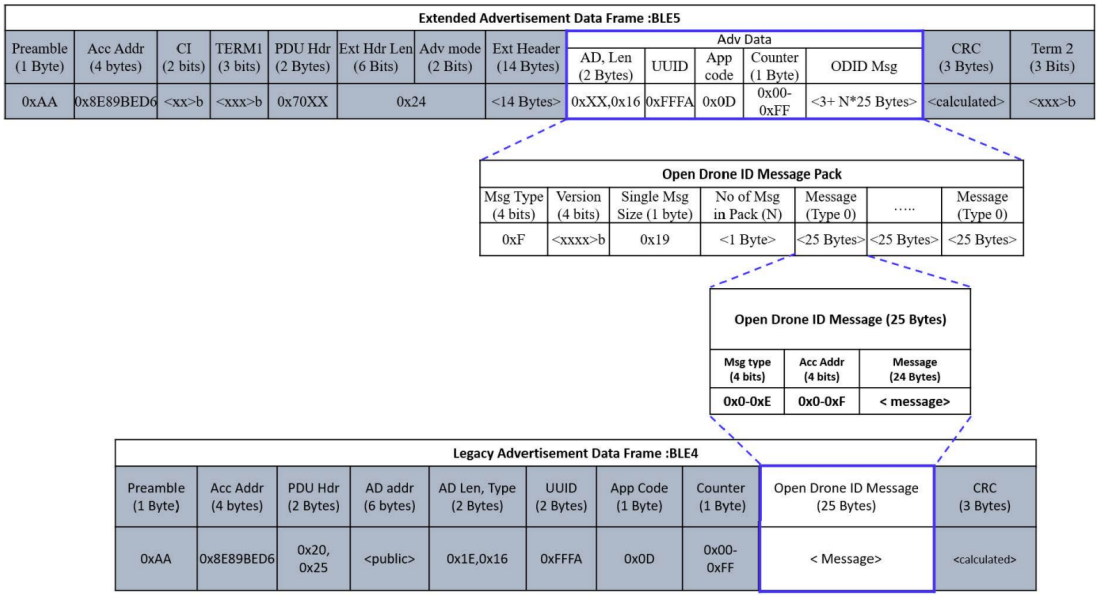

Figura 2.2: Diagrama de la trama del Open Drone ID transmitidos en BLE [14]

2.1.4. Spring Boot

Spring Boot es un *framework* que simplifica la construcción de aplicaciones Java con el ecosistema Spring mediante *auto-configuración*, dependencias *starter*. Permite exponer rápidamente una API REST con anotaciones como `@RestController`, integrando seguridad, métricas y tolerancia a fallos con configuración mínima [15].

Figura 2.3: Logotipo de Spring Boot [15]

2.1.5. Java Persistence API (JPA) y Spring Data JPA

JPA es la especificación de mapeo objeto–relacional estándar para Java, mientras que *Spring Data JPA* aporta repositorios genéricos que generan consultas

CRUD de forma automática [16, 17]. Esto permite definir una entidad y obtener de inmediato métodos como `findAll()` o `save()` sin escribir SQL. Acelera el desarrollo, favorece el *clean code*. Además, permite cambiar de motor SQL sin modificar la lógica de la aplicación.

2.1.6. Base de datos H2

H2 es una base de datos relacional embebida y, opcionalmente, persistente, con compatibilidad SQL-92 [18]. Se inicia dentro del mismo `jar` del *backend* y elimina la necesidad de instalar un servidor externo, mejorando así la portabilidad del TFG.

2.1.7. HTTP REST y JSON

La comunicación entre cliente y servidor se basa en *HTTP/1.1* [19] con métodos `POST` y `GET` y códigos de estado estándar. El cuerpo de los mensajes se serializa en *JSON*, formato de texto ligero ampliamente soportado tanto en Dart (`dart:convert`) como en Spring (Jackson). Este enfoque asegura la separación en bloques y facilita la depuración con herramientas como Postman.

2.1.8. Postman

Postman es una plataforma para enviar peticiones HTTP [20]. Se ha utilizado para comprobar el buen funcionamiento del *backend*.

Figura 2.4: Logotipo de Postman [20].

2.1.9. GitHub

La plataforma GitHub se ha consolidado como una herramienta indispensable en el diseño, desarrollo y gestión de proyectos tecnológicos, facilitando la colaboración y el control de versiones de manera eficiente. Para este proyecto, GitHub se ha utilizado para el control de las versiones en la producción del sistema. El repositorio de GitHub se encuentra en `https://github.com/igarrastazu/drone_by_drone`.

Figura 2.5: Logotipo de GitHub [21].

2.1.10. Visual Studio

Para el desarrollo del *frontend* de la aplicación, se seleccionó Visual Studio Code (VS Code) como el entorno de desarrollo integrado (IDE). Esta elección se tomó por su comodidad y facilidad.

VS Code ofrece un ecosistema robusto y extensible que se alinea perfectamente con los requisitos del proyecto:

- **Soporte extenso para Dart y Flutter**
- **Editor de texto potente**: incluye características como resaltado de sintaxis, múltiples cursores, y formateo automático.
- **Integración con GitHub**
- **Terminal integrado**

2.1.11. IntelliJ

Para el desarrollo del *backend* de la aplicación, se optó por IntelliJ IDEA de JetBrains. Esta elección se basó en su robusto soporte para el desarrollo con Java y Spring Boot, así como en sus potentes herramientas que facilitan la gestión de proyectos complejos y la optimización del código.

Capítulo 3

Estado de la cuestión

El problema encontrado por la empresa Drone by Drone [1] es que el alcance Wi-Fi / Bluetooth de las balizas no es suficiente al trabajar en un área saturada de drones.

Partiendo de este problema, la idea propuesta es el desarrollo de un sistema que sea capaz de almacenar la localización del dron detectado por el dispositivo y de enviar esa localización a todos los dispositivos que tengan acceso al sistema. Este proyecto puede ayudar a que se logre aumentar el alcance de detección de drones, especialmente drones dentro de la misma organización que utilice este sistema con varios dispositivos.

3.1. Punto de vista técnico

En 2016 la EASA definió que los sistemas deberían transmitir el registro del operador, el tipo de UAS, y su localización y altura. En cuanto al tipo de UAS, se diferenciarían en la categoría abierta y la específica/cerrada. Los drones de esta categoría específica requieren autorización especial de la National Airspace Agency, y tienen características especiales como la posibilidad del transporte de personas y material peligroso [22].

La categoría abierta (la mayoría de los drones utilizados hoy en día) se dividee en una serie de tipos de UAS en función de su peso. Todos los drones, tanto de la categoría abierta como de la específica, están obligados a utilizar DRID (Direct Remote ID), el cual transmite la información exigida por la EASA por Wi-Fi y/o Bluetooth (dependiendo de lo que el fabricante prefiera). Esta información se emite en tiempo real en *broadcast*, por lo que cualquier dispositivo dentro del alcance

puede recibirla si tiene el software adecuado para la interpretación de los mensajes y una antena Wi-Fi/receptor Bluetooth.

Debe transmitir obligatoriamente el ID del dron (número de serie del UAV o de la sesión ID), la latitud y longitud, la altura y la velocidad del dron [23]. Estos datos serían los guardados en la base de datos, con especial importancia en la latitud y longitud del dron para poder geolocalizarlo.

3.2. Estado actual de los sistemas de detección de drones

Existen muchas aplicaciones que utilizan esta información transmitida para localizar los drones en el alcance Wi-Fi/Bluetooth. Un ejemplo de estas aplicaciones se encuentra en Google Play: Drone Scanner. Drone Scanner es una aplicación proporcionada por Dronetag para la localización de drones con Open Drone ID. También permite configurar sus balizas. En la Figura 3.1 se puede ver cómo es esta aplicación.

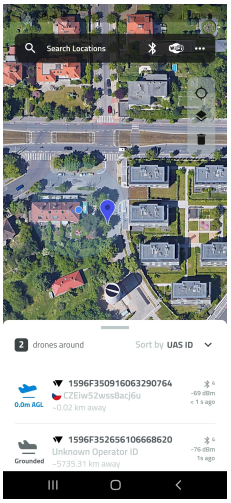

Figura 3.1: Captura de pantalla de Drone Scanner [13].

Si buscamos un ejemplo más similar a lo que esta aplicación propone, nos encontramos con NRID (Network Remote ID). NRID es un protocolo que contiene una baliza que se conecta al propio dron. Esta baliza comparte esta información a una red externa (en lugar de transmitirlo en broadcast por Wi-Fi/Bluetooth). Normalmente se utiliza LTE 4G/5G/XG para transmitir estos datos [24]. Esta red

externa almacena esta información, y para acceder a esta información debes tener conexión a Internet.

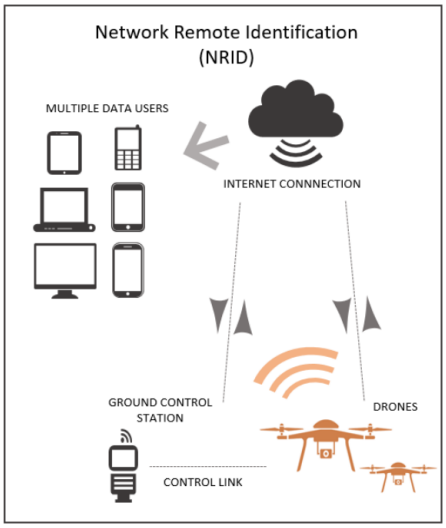

Figura 3.2: Funcionamiento NRID [24]

Esta tecnología se asemeja a lo que esta aplicación quiere conseguir. Normalmente, sólo se les añade esta tecnología a drones con supervisión avanzada por su coste. El sistema propuesto pretende utilizar DRID y su alcance Wi-Fi/Bluetooth. El propio dispositivo que haya localizado el dron será el encargado de subir la información de este al servidor, y no la propia baliza del dron.

Capítulo 4

Definición del Trabajo

4.1. Justificación

En los últimos años, la regulación internacional ha comenzado a exigir sistemas de identificación remota en drones para mejorar la seguridad del espacio aéreo. Esta necesidad, junto con el auge de soluciones *open-source* como Open Drone ID, ha motivado la creación de muchas aplicaciones para la localización de estos drones.

Este trabajo de fin de grado responde a esa necesidad mediante el desarrollo de un sistema que permita detectar drones que transmiten su posición a través de balizas compatibles con el estándar Open Drone ID, visualizar su localización en tiempo real y almacenar sus datos para su uso mediante un sistema colaborativo.

Figura 4.1: Logotipo de la empresa Drone by Drone [1].

Además, la elección del tema está motivada por su alineación con las actividades de la empresa *Drone by Drone 4.1* y los problemas que han tenido con el uso de otras aplicaciones en el mercado. Esto lleva a un sistema que ofrezca lo siguiente:

Aumentar el radio de alcance de las tecnologías Wi-Fi/Bluetooth

Al trabajar con un número alto de drones o compartir el espacio aéreo con drones de otras empresas, el alcance de estas balizas puede no ser suficiente.

La tecnología Wi-Fi utiliza frecuencias en el rango de gigahercios. El rango de cobertura de algunas de las antenas Wi-Fi no excede los 300 metros, aunque alguna antena pueda superar 1 kilómetro de distancia [22].

El Bluetooth ofrece una conexión segura con un consumo bajo de batería. El ancho de banda puede alcanzar los 650 kbps. Esta tecnología suele alcanzar los 200 metros [22], pero puede verse disminuida en enorme medida por cuestiones del entorno.

Comunicación entre diferentes usuarios

La creación de este sistema puede facilitar la creación de una red de drones, en la que todos los dispositivos móviles puedan verlos simultáneamente. Esto puede facilitar la comunicación entre los trabajadores de una empresa o un grupo de usuarios con una gran cantidad de drones.

Minimizar posibles accidentes

Aumentando el área de alcance en el que se pueden observar los drones puede disminuir numerosos accidentes con los drones gracias a la supervisión de un equipo más grande que pueda acceder desde un alcance mayor y ver la localización de todos los drones en uso.

Para disminuir los accidentes con los drones en lo que más necesitamos centrarnos es en el correcto seguimiento y vigilancia de los drones. Es muy importante también el conocimiento de zonas restringidas y procedimientos de seguridad básicos [4]. La necesidad de tener el acceso sencillo e intuitivo a esta información de manera dinámica ha motivado también la creación de este sistema.

Incentivar el uso de drones

Los drones pueden facilitar el trabajo en numerosos sectores como la construcción o el mantenimiento de infraestructuras. El desarrollo de este sistema puede incentivar el uso de esta tecnología en sectores en los que por este límite de alcance no sea factible o en los que una cantidad elevada de drones sea necesaria.

La empresa Drone by Drone ha proporcionado asesoramiento técnico y equipamiento (una baliza Dronetag Beacon) para el desarrollo y validación del sistema. Esta colaboración refuerza el componente aplicado del trabajo, ya que parte de una necesidad existente en el sector y busca ofrecer una solución funcional y extensible.

A nivel académico, el proyecto supone una oportunidad para aplicar conocimientos de desarrollo móvil, *backend*, comunicación BLE y bases de datos en un contexto real y actual.

Por tanto, este proyecto no solo me permite aplicar y consolidar los conocimientos adquiridos durante el grado, sino que también contribuye a un problema técnico y actual, con posibilidades reales de uso y extensión en entornos profesionales relacionados con el control de espacio aéreo y operaciones con drones.

4.2. Motivación del proyecto

El desarrollo de este proyecto surge del interés personal por el campo de los sistemas distribuidos, las tecnologías móviles y su aplicación en entornos reales. A lo largo del Grado de Ingeniería en Tecnologías de Telecomunicación he tenido la oportunidad de adquirir conocimientos en el desarrollo de aplicaciones (en especial del lado servidor), y consideré que un proyecto con arquitectura cliente-servidor y comunicación en tiempo real representaba un reto técnico interesante y formativo.

Además, el creciente protagonismo de los drones en ámbitos como la seguridad, la logística o la inspección industrial ha despertado en mí un interés especial por las tecnologías asociadas a estos dispositivos. Lo cierto es que considero que estos dispositivos tienen muchas posibilidades, que también vienen acompañadas de riesgos y peligros, los cuales me gustaría ayudar a reducir. La posibilidad de trabajar con una baliza real compatible con el estándar Open Drone ID, cedida por la empresa colaboradora *Drone by Drone* me ofrecía una oportunidad única para aplicar conocimientos teóricos en un entorno real y profesional.

En conjunto, este proyecto me ha permitido integrar conocimientos adquiridos durante la carrera, explorar tecnologías emergentes que no conocía como Flutter y participar en un proceso de desarrollo completo con un fuerte componente práctico y, sobretodo, una sensación de aportar al mercado laboral y de que mi proyecto tenga una utilidad real.

4.3. Objetivos

El proyecto tiene como finalidad el desarrollo de una aplicación funcional capaz de detectar, interpretar y visualizar datos emitidos por drones mediante balizas de identificación remota siguiendo el estándar Open Drone ID y compartirlo entre el resto de usuarios del sistema. Para lograrlo, se han definido una serie de objetivos que guían el trabajo tanto desde el punto de vista técnico como organizativo, y que permiten estructurar las distintas fases del desarrollo.

4.3.1. Objetivo general

El objetivo general es el desarrollo de un sistema que permita recibir, visualizar y registrar la posición de drones equipados con balizas Open Drone ID, almacenando los datos obtenidos en un sistema *backend* centralizado y base de datos relacional y pudiendo acceder a ellos a través de peticiones en el *frontend*.

4.3.2. Objetivos específicos

1. Diseñar e implementar una interfaz de usuario que permita registrar e iniciar sesión con un usuario y que escanee tramas BLE emitidas por una baliza con mensajes Open Drone ID.

2. Lograr que el sistema reciba las localizaciones y persista los datos en una base de datos relacional.

3. Modelar y construir una base de datos que permita almacenar información básica del dron (posición, hora registrado e ID del dron) de forma estructurada y consultable, garantizando la integridad de los datos.

4. Implementar esta interfaz de usuario y esta base de datos siguiendo un modelo por capas que asegure una alta cohesión intracapa y un bajo acoplamiento entre ellas.

4.4. Metodología

Para la realización de este proyecto se ha elegido seguir una **metodología mixta**, que combina elementos del enfoque tradicional en cascada (*Waterfall*) con prácticas del modelo ágil (*Agile*).

El modelo en cascada, también conocido como enfoque tradicional, se caracteriza por su estructura secuencial y bien definida, en la que cada fase (análisis,

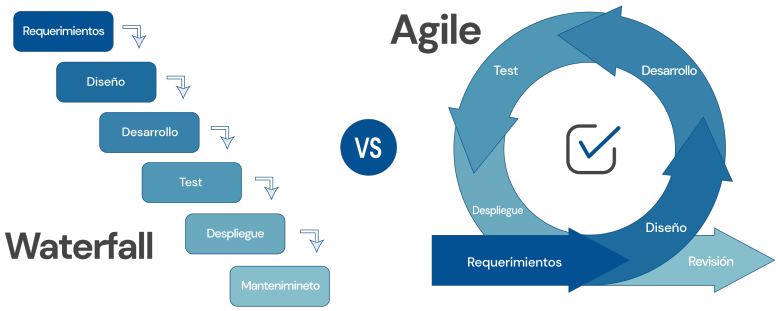

Figura 4.2: Metodología en cascada y metodología ágil [25]

diseño, implementación, pruebas, etc.) debe completarse completamente antes de pasar a la siguiente. Esta aproximación resulta especialmente útil para la planificación inicial, la definición de requisitos y la elaboración de documentación formal, aspectos fundamentales en un Trabajo de Fin de Grado.

Por otro lado, el enfoque ágil promueve la iteración continua, la entrega incremental de funcionalidades y la flexibilidad frente a cambios. Durante el desarrollo práctico de la aplicación, se han adoptado prácticas propias de este enfoque, como:

- Descomposición del trabajo en tareas cortas y manejables.
- Revisión y mejora iterativa de la interfaz y la lógica del sistema.
- Pruebas frecuentes sobre versiones parciales del software.
- Priorización de funcionalidades mínimas viables (MVP).

Se puede observar en la Figura 4.2 el esquema de trabajo de ambas metodologías.

Esta combinación ha permitido mantener una planificación clara y documentada, acorde con los requisitos académicos, al mismo tiempo que favorecía un desarrollo flexible y adaptable a medida que surgían nuevas necesidades o limitaciones técnicas.

Para la elaboración del sistema se dividió el trabajo en diferentes fases independientes. Ciertas de estas fases se realizaron a la vez para poder agilizar el proceso. Se realizaron *sprints* de entre 2 o 3 semanas. Al finalizar cada *sprint*, se realizó un sprint review con el director del TFG y con la empresa Drone by Drone para garantizar el buen avance del proyecto. También hubo una reunión al principio

de cada *sprint*, con la intención de facilitar la organización necesaria para llevar a cabo el trabajo.

4.5. Planificación y estimación económica

Fue necesaria una presentación temporal de las actividades a realizar y una estimación del costo de desarrollo.

Cronograma (Plan de Trabajo)

Para llevar a cabo el desarrollo del proyecto, se ha elaborado una planificación temporal basada en una división por fases, cada una centrada en una actividad clave del TFG. Estas fases incluyen desde la investigación y análisis inicial hasta el desarrollo de la aplicación, además de la redacción de la memoria del proyecto.

La planificación ha sido organizada en función del calendario académico. A continuación, se presenta la tabla resumen con las fases y sus fechas estimadas:

Actividad	Fecha de Inicio	Duración (días)	Fecha Final
Fase 1: Planificación	15/12/2024	30	14/01/2025
Fase 2: Aprendizaje de Tecnologías	14/01/2025	20	03/02/2025
Flutter	14/01/2025	10	24/01/2025
Java (*Backend*)	24/01/2025	5	29/01/2025
Acceder ubicación baliza	29/01/2025	5	03/02/2025
Fase 3: Modelado del sistema	03/02/2025	30	05/03/2025
Diagrama de casos de uso	03/02/2025	5	08/02/2025
Diagramas de secuencias	08/02/2025	5	13/02/2025
Fase 4: Desarrollo de la primera versión	03/02/2025	75	19/04/2025
Fase 5: Recogida de pruebas y análisis	19/04/2025	5	24/04/2025
Fase 6: Desarrollo de la versión final	24/04/2025	40	03/06/2025
Fase 7: Redacción y presentación del TFG	14/01/2025	140	03/06/2025

Tabla 4.1: Planificación temporal del proyecto

Se hizo también un diagrama de Gantt en Excel. Este expone cada una de las fases del proyecto y su duración estimada.

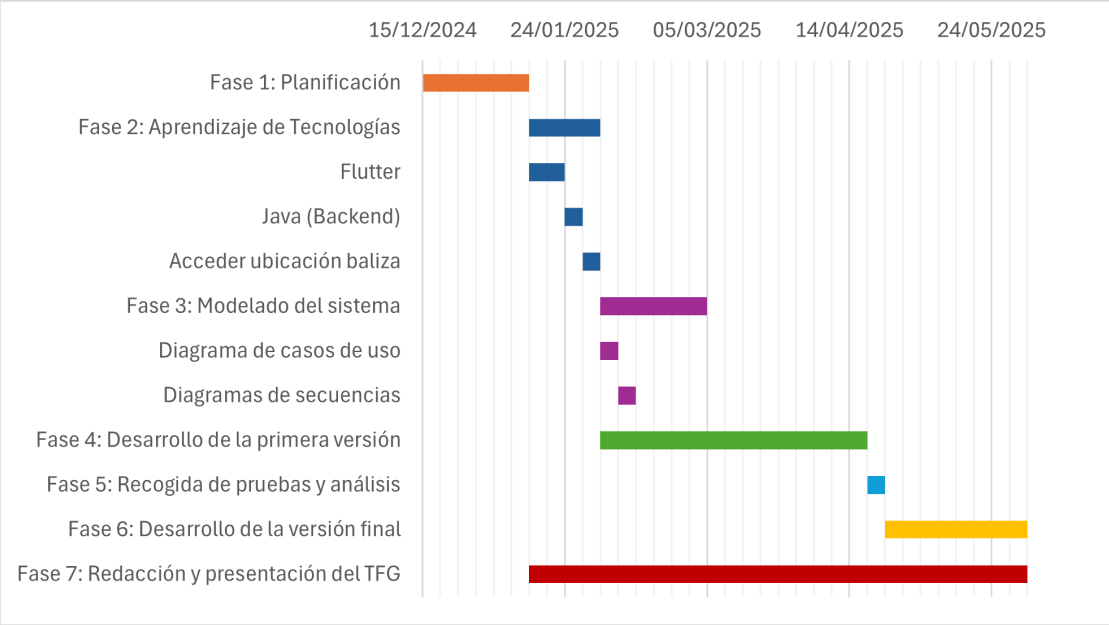

Figura 4.3: Diagrama de Gantt.

Las fases realizadas fueron las siguientes:

- **Fase 1 Planificación:** en esta primera etapa se organizó la metodología del proyecto. También se situó el contexto del proyecto, los recursos a emplear y los objetivos y motivaciones que han llevado a hacer el proyecto.
- **Fase 2 Aprendizaje de tecnologías:** en esta fase se aprendió los conocimientos que fueron necesarios para la realización del proyecto. Esta fase fue crucial para comprobar si era posible cumplir con los objetivos propuestos, especialmente para la comunicación con el dron.
- **Fase 3 Modelado del sistema:** esta etapa consiste en estructurar los diagramas UML y todos los modelos necesarios antes de comenzar con el desarrollo del sistema.
- **Fase 4 Desarrollo de la primera versión:** en esta fase se realizó la primera versión del sistema.
- **Fase 5 Recogida de pruebas y análisis:** durante la recogida de pruebas y análisis se comprobó si el funcionamiento de esta primera versión cumplía los requisitos necesarios.
- **Fase 6 Desarrollo de la versión final:** durante esta fase se perfeccionó

la primera versión con el *feedback* recibido de Drone by Drone y el tutor del TFG para lograr los objetivos propuestos en la planificación del proyecto.

- **Fase 7 Redacción y presentación del TFG:** la última fase se trata de redactar en LaTeX la memoria del TFG y preparar la presentación final. Esta fase se hizo al mismo tiempo el resto de fases para lograr explicar en detalle cada uno de los procesos realizados.

4.5.1. Estimación económica

El desarrollo del proyecto no ha supuesto un coste económico directo, ya que no se ha empleado infraestructura en la nube ni licencias comerciales. No obstante, se detallan a continuación los recursos materiales utilizados, indicando su origen y coste estimado en caso de que se hubiera requerido adquirirlos.

- **Ordenador portátil HP Pavilion x360**: equipo personal empleado tanto para el desarrollo del software como para la redacción de la documentación.
  Coste: 899.01 €
- **Baliza Dronetag Beacon**: dispositivo necesario para la detección de drones con Remote ID conforme al estándar Open Drone ID. Este dispositivo ha sido cedido por la empresa colaboradora Drone by Drone exclusivamente para la realización del proyecto.
  Coste: 199 €

La estimación económica total del proyecto, si se hubiera requerido adquirir todos los recursos empleados, sería de aproximadamente:

Total: 1098.01 €

Todo el entorno de desarrollo empleado es completamente gratuito (Flutter, Spring Boot, base de datos H2, etc.), lo cual ha facilitado la realización del proyecto sin necesidad de inversión adicional.

4.6. Requisitos

Este proyecto pretende resolver una serie de requisitos para su correcto funcionamiento y para poder cumplir con los problemas encontrados que han motivado la elaboración de esta aplicación. El objetivo principal es el desarrollo de una aplicación que facilite la localización de drones más allá del alcance WiFi/Bluetooth gracias a la conexión con otros dispositivos móviles, algo similar al funcionamiento del Apple AirTag.

4.6.1. Requisitos funcionales

Los requisitos funcionales describen lo que un sistema debe hacer. Estos se enfocan en la funcionalidad, el comportamiento y las acciones específicas del sistema.

1. El sistema debe localizar satisfactoriamente los drones alcanzados con el rango de la baliza utilizando la tecnología BLE (Bluetooth Low Energy) y Wifi.

2. El sistema debe ser capaz de enviar a la base de datos la localización de los drones localizados con BLE y Wifi.

3. El sistema debe ser capaz de recibir los drones de la base de datos.

4. Se debe diferenciar qué drones vienen de la base de datos y qué drones el dispositivo los localiza por Bluetooth y por Wi-Fi.

4.6.2. Requisitos no funcionales

Los requisitos no funcionales describen cómo el sistema debe funcionar. Se centran en atributos de calidad como rendimiento, seguridad y usabilidad.

1. La integración de estas localizaciones debe ser rápida y dinámica, para que el usuario pueda verlas en tiempo real.

2. Tanto el *backend* como el *frontend* deben ser robustas y poder recibir una cantidad elevada de drones detectados sin saturarse.

3. El sistema debe realizarse de manera escalable. Esto significa que el sistema debe de estar preparado para ser actualizable de manera sencilla si el número de drones o usuarios aumenta.

Capítulo 5

Sistema desarrollado

5.1. Análisis del Sistema

El análisis del sistema describe la descomposición de los componentes principales de la aplicación y la interacción entre ellos.

5.1.1. Arquitectura General del Sistema

La arquitectura del sistema sigue el modelo cliente-servidor. Está compuesta por cuatro elementos principales que se pueden observar en la Figura5.1. Estas partes son las siguientes:

1. **La baliza Dronetag Beacon** instalada en el dron, que emite paquetes BLE con su identificación y telemetría.

2. **La aplicación móvil Flutter (*frontend*)**, que escanea los anuncios BLE, parsea la trama según la especificación Open Drone ID y envía la información al servidor mediante peticiones HTTP REST en formato JSON. También actúa como la capa de interfaz de usuario (cliente)

3. **El *backend* Spring Boot**, organizado en capas (controlador, servicio, repositorio y modelo) y responsable de recibir las peticiones, aplicar la lógica de negocio. Además es el encargado de acceder a la persistencia.

4. **La base de datos relacional H2**, donde los datos se almacenan a través de Spring Data JPA.

Como se puede observar en la Figura 5.1, el flujo completo de datos es: Baliza → App Flutter → API REST → Base de datos, garantizando una separación clara entre la captura de la señal, el procesamiento en el cliente, la lógica del servidor y el almacenamiento persistente.

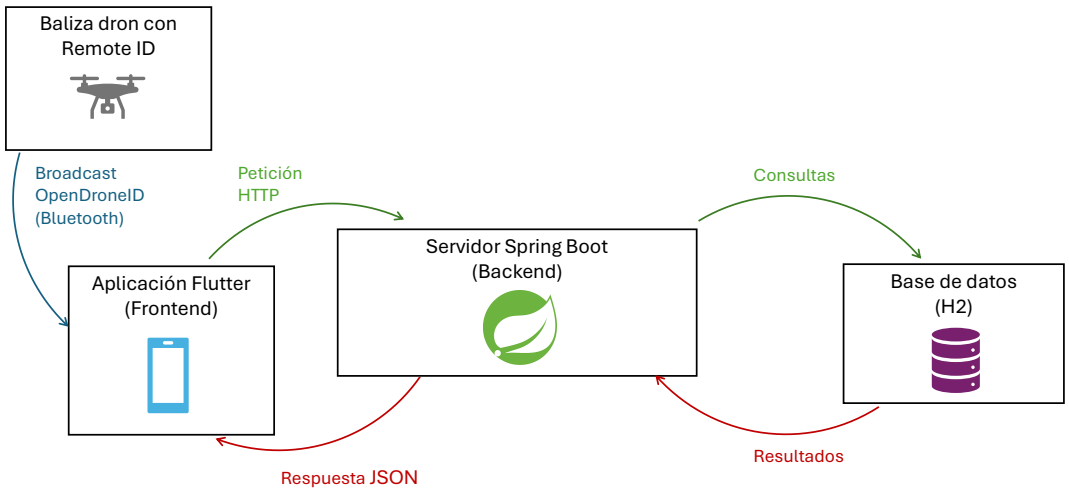

Figura 5.1: Diagrama de caja negra.

Capa de presentación (*frontend*)

Es la capa encargada de la interfaz visual. Gracias a esta capa, el usuario puede interactuar con el sistema. Permite a los usuarios registrarse, iniciar sesión y visualizar la localización de drones en tiempo real (tanto los que tiene a su alcance como los que están almacenados en la base de datos). Además, envía los drones localizados a la base de datos a través de peticiones. Estará implementado únicamente en el sistema operativo Android. Está desarrollada con Flutter y cuenta con una API de Google para la gestión del mapa.

Capa de acceso a datos (*backend*)

Es la capa encargada de la manipulación y el procesamiento de los datos. Está desarrollado en Java con Spring Boot. Spring Boot es un *framework* de desarrollo en Java. Se basa en el ecosistema Spring, y elimina la necesidad de realizar configuraciones manuales complejas mediante el uso de convenciones ya programadas [15]. En esta aplicación se utiliza para desarrollar APIs REST de forma ágil y modular.

El mapeo en la base de datos sigue la API JPA (Java Persistence API), que define tablas de base de datos como objetos en Java [16]. Está gestionado con

Spring Data JPA, que es un módulo del proyecto Spring Data que facilita el uso de JPA [17].

Para el almacenamiento de datos se ha optado por utilizar H2 por su simplicidad. En un entorno de producción, el sistema podría adaptarse fácilmente a otras bases de datos relacionales como PostgreSQL o MySQL, sin apenas modificar el código, gracias a la abstracción que proporciona Spring Data JPA.

5.2. Diseño

Antes de la elaboración del sistema fue realizado el diseño del mismo. Este se puede dividir en las secciones: diseño de la comunicación cliente-servidor, diseño de la base de datos, diseño del *frontend* y diseño del *backend*.

5.2.1. Diseño de la comunicación cliente-servidor

La aplicación sigue un diseño cliente-servidor. La API se puede separar en dos partes: las rutas de la API de usuarios y las rutas de la API de drones.

Rutas de la API de usuarios

Estas rutas están dedicadas a manejar y manipular la información del usuario. Trabaja con una *cookie* en la cabecera de la petición, que permite modificar la información del usuario sin necesidad de introducir su contraseña en cada petición 5.1.

Rutas de la API de drones

Estas rutas, en cambio, están reservadas solamente para subir y recibir información de los drones que el dispositivo detecta. También trabaja con la *cookie* de usuario en la cabecera de la petición para autenticar al usuario. Esto habilita que los usuarios no autenticados no sean capaces de recibir drones, pero sí de enviar los drones que el dispositivo detecta. 5.2.

5.2.2. Diseño de la base de datos

La base de datos es una base de datos relacional H2 dividida en las tablas Drone, Token y AppUser. Se puede observar el diagrama de la base de datos en la Figura 5.2.

Método	Ruta	Descripción	Respuestas
POST	/api/users	Registra a un usuario con un nombre, email, rol y contraseña.	201 - creado 409 - conflicto (usuario ya registrado) 500 - error del servidor.
POST	/api/users/me /session	Inicia sesión con un email y una contraseña si es correcto.	200 - ok (cuenta correcta) 401 - no autorizado.
DELETE	/api/users/me /session	Cierra sesión con la información de las credenciales de las cookies.	204 - solicitud correcta 401 - no autorizado.
GET	/api/users/me	Devuelve los datos del usuario (nombre, email y rol).	200 - ok (datos enviados) 401 - no autorizado.
GET	/api/users/all	Devuelve los datos de todos los usuarios (nombre, email y rol) (solo admin).	200 - ok (datos enviados) 401 - no autorizado.
DELETE	/api/users/me	Borra la cuenta usando las cookies de autenticación.	204 - borrado 401 - no autorizado.

Tabla 5.1: Resumen de rutas de la API de usuarios

Tabla AppUser

Es la tabla en la que se guarda la información del usuario:

- **ID**: es un valor generado automáticamente para identificar al usuario. Es la *primary key*.
- **Nombre**: es el nombre del usuario.
- **Email**: es el email del usuario. Debe ser único en la base de datos y nunca podrá ser nulo.
- **Contraseña**: la contraseña del usuario sirve para poder iniciar sesión. Se almacena cifrada mediante una función *hash* por seguridad del usuario. La petición para registrar un usuario limita la contraseña a una contraseña con al menos una mayúscula, una minúscula, un número, y de una longitud mayor que 7.
- **Rol**: diferencia a usuarios administradores y usuarios normales. Aunque en la versión actual de la aplicación no sea de utilidad, se podrá definir contenido exclusivo a administradores como la posibilidad de que los drones subidos

Método	Ruta	Descripción	Respuestas
POST	/api/drone	Sube los drones detectados por el dispositivo e incluye en el cuerpo de la respuesta todos los drones de la base de datos (salvo los que el usuario ha mandado en el cuerpo de la petición).	200 - ok 204 - no content (respuesta sin drones) 203 - non authoritative information (usuario sin sesión) 401 - no autorizado (cookie no válida) 409 - error de la petición. 400 - bad request
GET	/api/drone /all	Recibe todos los drones de la base de datos sin enviar ningún dron.	200 - ok 401 - no autorizado (cookie no válida).

Tabla 5.2: Resumen de rutas de la API de drones

por ellos sólo puedan acceder ellos mismos u otros usuarios elegidos por ellos en un futuro.

Tabla Token

Guarda el token de inicio de sesión del usuario.

- **ID**: es un valor generado automáticamente para identificar al *token*. Es el valor que se almacena en los *headers* de la petición HTTP como una *cookie* de sesión.
- **ID del usuario**: el ID del usuario enlazado al token.

Tabla Drone

La tabla de los drones es la más extensa de la base de datos. Esta consta de las siguientes columnas:

- **ID**: es un valor generado automáticamente para identificar al dron. Es la *primary key*.
- **Remote ID**: es el ID del dron. Normalmente es el *serial number* (número de serie) del dron. Es único en la base de datos y no puede ser nulo.

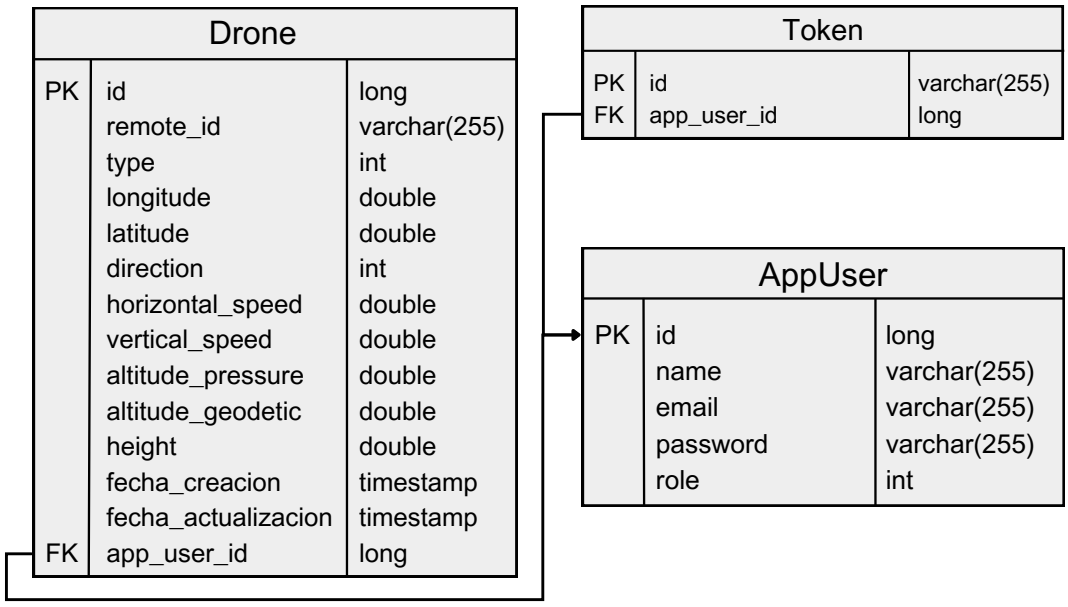

Figura 5.2: Diagrama de base de datos

- **Tipo de Remote ID**: existen varios tipos de Remote ID. Estos tipos deben ser indicados en la base de datos. Se encuentran en el Código 5.1.
- **Longitud y latitud:** es la posición del dron.
- **Información del dron:** son datos del dron con valor meramente informativo. La aplicación registra estos valores de los mensajes del dron:
  - Dirección.
  - Velocidad horizontal vertical.
  - Presión del aire en el dron.
  - Altura geométrica.
  - Altura del dron (respecto al nivel del mar)
- **Fechas de creación y de actualización**: se generan automáticamente. La fecha de actualización es especialmente útil para eliminar drones de la base de datos que lleven más tiempo sin recibir nuevos datos que el tiempo máximo permitido.
- **ID del usuario**: id del usuario que ha visto ese dron por última vez.

5.2.3. Diseño del *frontend*

El *frontend* del sistema se ha organizado en diferentes módulos o componentes, cada uno de ellos con una responsabilidad específica. Esta división modular facilita la mantenibilidad del código, mejora la legibilidad y permite una evolución más controlada del sistema. En el Código 5.2 se puede ver la ruta de los archivos del *frontend*. A continuación, se describen los principales módulos:

Páginas

El módulo de páginas contiene las diferentes vistas principales de la aplicación móvil desarrollada en Flutter. Cada clase representa una pantalla completa, generalmente asociada a una ruta de navegación, y está compuesta por uno o varios *widgets* que estructuran la interfaz visual del sistema.

Estas páginas encapsulan tanto la presentación como parte de la lógica de interacción con el usuario. A continuación, se detallan las principales pantallas implementadas en el sistema:

- **Pantalla principal (mapa) (/map):** es la vista inicial de la aplicación. Su principal función es mostrar el mapa interactivo en el que se representan las ubicaciones de los drones detectados. Utiliza el paquete `google_maps_flutter` y superpone marcadores personalizados que muestran información relevante sobre cada dron (identificador, posición, altitud, etc.). Desde esta pantalla también se puede acceder a la información del usuario [1]. Además, puedes pulsar sobre uno de los drones representados en el mapa. Esto muestra información más específica sobre el dispositivo, como la velocidad, altitud, fecha de recepción de la última señal, etc.

- **Pantalla de autenticación (/auth):** es la primera pantalla que ves al abrir la aplicación. Sirve para obtener la *cookie* de sesión, que permitirá acceder a los drones de la base de datos (tanto registrándote como iniciando sesión). [2] También puedes ir a la página del mapa directamente si le das al botón de `Continuar en local` o si tu dispositivo ha guardado ya una *cookie* de sesión válida de otro inicio de sesión, ya que al iniciar la aplicación esto es lo primero que se comprueba.

- **Pantalla de información del usuario (/user):** se accede a ella pulsando el icono de usuario en el mapa siendo un usuario registrado. Desde esta

[1] Si haces *click* en el mapa a este icono y no hay una sesión activa te llevará a la pantalla de inicio de sesión.

[2] Como se puede observar en el diagrama de casos de uso (Figura 5.6), el usuario sin registrar puede ver los drones locales, pero no los subidos a la base de datos.

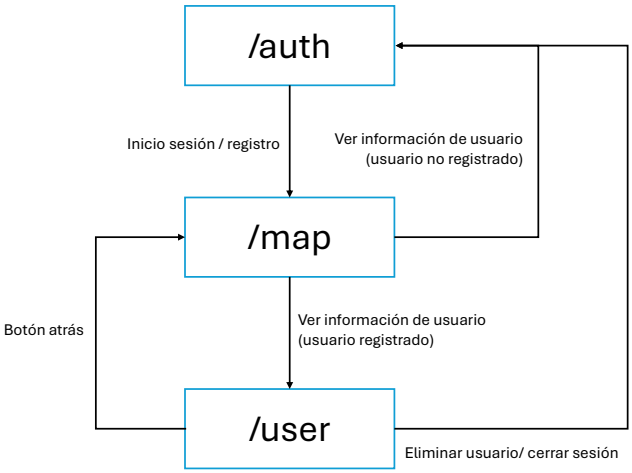

Figura 5.3: Diagrama de Navegación

pantalla puedes ver la información del usuario, eliminar el usuario y cerrar sesión. Estos dos últimos botones te llevarán de nuevo a la pantalla de inicio de sesión y registro.

Cada una de estas pantallas se encuentra estructurada en archivos independientes dentro de la carpeta `lib/pages/`, y utiliza *widgets* personalizados definidos en módulos auxiliares. La navegación entre pantallas se gestiona mediante el sistema de rutas de Flutter a través de la librería `go_router`, y se puede ver en la Figura 5.3.

Servicios

El módulo de servicios del *frontend* contiene clases que encapsulan la lógica de acceso a datos y funcionalidades específicas que no pertenecen directamente a la interfaz de usuario. Su principal función es actuar como capa intermedia entre los *widgets* visuales y la información, ya sea una API REST, una señal Bluetooth o almacenamiento local. Esta separación permite desacoplar la lógica del negocio de la presentación, lo que favorece la reutilización, la prueba y la mantenibilidad del código.

Entre los servicios implementados en la aplicación destacan los siguientes:

- **Servicio de red (API Service):** se encarga de gestionar todas las peticiones HTTP al *backend*. Utiliza la librería `http`. Se divide en:

  - **API Service Usuarios**: hace las peticiones relacionadas con el usuario (las peticiones de la Figura 5.1). También controla la *cookie* de sesión y la almacena localmente.
  - **API Service Drones**: hace las peticiones relacionadas con los drones (las peticiones de la Figura 5.2).

- **Servicio de Bluetooth (BLE Service):** gestiona el escaneo y recepción de señales BLE compatibles con el protocolo Open Drone ID.
- **Proxy Bluetooth:** se enfoca en imitar el comportamiento de la recepción de mensajes de Open Drone ID por Bluetooth. Ha sido necesario por la disposición limitada de la baliza.

Cada servicio está estructurado como una clase separada dentro del directorio `lib/servicios/` (se puede ver la ruta de los archivos en el Código 5.2).

Útil

El módulo `utils` agrupa funciones auxiliares, constantes y clases estáticas que no pertenecen a una pantalla ni a un servicio concreto, pero que se utilizan de forma recurrente. En este sistema se encuentran las clases que formatean los mensajes de Open Drone ID

- **MessageDroneBasicId:** formatea el mensaje que contiene el remote ID del dron. También incluye la función `getIdValue` que traduce el mensaje Basic ID de Open Drone ID en un formato válido para el backend del sistema.
- **MessageDroneBasicIdType:** enumeración que identifica el tipo de remote ID del dron. Si no se sabe el remote ID se registrará la MAC del dron.

```
enum MessageDroneBasicIdType {
SERIAL_NUMBER,
CAA_REGISTRATION_ID,
UTM_ASSIGNED_ID,
SPECIFIC_SESSION_ID,
MAC_ADDRESS
}
```

Código 5.1: Tipos de Remote ID.

- **MessageDroneLocation:** formatea el mensaje de la localización del dron en un formato válido para el *backend* del sistema.

Assets

La carpeta `assets` contiene los recursos estáticos utilizados por la aplicación, tales como imágenes, archivos de texto, configuraciones o datos de prueba. Estos elementos no forman parte del código fuente directamente, pero son necesarios para la presentación visual de la interfaz o el funcionamiento de determinados módulos.

En Flutter, los recursos ubicados en la carpeta `assets/` deben declararse explícitamente en el archivo `pubspec.yaml` [3] para que puedan ser empaquetados e incluidos correctamente durante la compilación de la aplicación. Esto permite acceder a dichos recursos mediante rutas relativas desde el código Dart, y garantiza su disponibilidad en tiempo de ejecución.

```
lib
+-- assets
|   +-- logo.png
+-- pages
|   +-- auth_page.dart
|   +-- login_page.dart
|   +-- map_page.dart
|   +-- register_page.dart
|   +-- user_info_page.dart
+-- servicios
|   +-- api_service_drone.dart
|   +-- api_service_user.dart
|   +-- flutter_odid_bluetooth_scanner.dart
|   +-- proxy_bluetooth_scanner.dart
|   +-- scanner.dart
+-- util
|   +-- message_drone_basic_id_type.dart
|   +-- message_drone_basic_id.dart
|   +-- message_drone_location.dart
+-- widgets
|   +-- custom_text_field.dart
+-- main.dart
```

Código 5.2: Ruta de archivos del *frontend*

[3] El archivo `pubspec.yaml` es un fichero fundamental cuya función principal es definir la configuración del proyecto, incluyendo las dependencias, los recursos estáticos y los metadatos del paquete. Gracias a este archivo, el gestor de paquetes `pub` puede instalar las bibliotecas necesarias y configurar el entorno de ejecución de forma automática.

5.2.4. Paleta de colores

Para asegurar una interfaz de usuario coherente y estéticamente agradable, se definió una paleta de colores específica para la aplicación. Esta selección se basó en los colores corporativos de la empresa Drone by Drone. Además, se tuvieron en cuenta principios de usabilidad y accesibilidad, buscando una combinación que facilitara la legibilidad y la distinción de elementos, a la vez que proporcionara una identidad visual clara al sistema.

Los colores principales y secundarios utilizados en la aplicación se detallan a continuación, junto con sus respectivos códigos hexadecimales para una replicación precisa:

- Gris claro (#B3B3B3)
- Gris oscuro (#5C5C5C)
- Negro (#25282D)
- Rojo (#C62922)
- Blanco sucio (#EEEEEE)

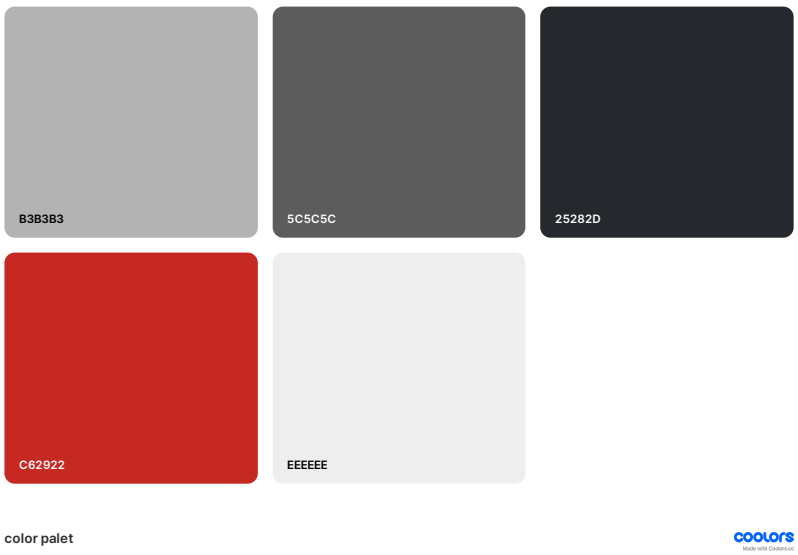

Figura 5.4: Paleta de colores (diagrama hecho con la herramienta Coolors).

5.2.5. Diseño del *backend*

El *backend* del sistema sigue el patrón MVC (Modelo-Vista-Controlador) con algunos cambios [4]. Divide el sistema en capas/módulos separados según su responsabilidad. Se puede ver la ruta de archivos en el Código 5.3.

```
com.icai.dronebydrone_backend
+-- controller
|   +-- DroneController
|   +-- UserController
+-- entity
|   +-- AppUser
|   +-- Drone
|   +-- Token
+-- model
|   +-- DroneRequest
|   +-- DroneResponse
|   +-- LoginRequest
|   +-- MessageDroneBasicIdType
|   +-- ProfileRequest
|   +-- ProfileResponse
|   +-- RegisterRequest
|   +-- Role
+-- repository
|   +-- AppUserRepository
|   +-- DroneRepository
+-- service
|   +-- DroneCleanupService
|   +-- DroneService
|   +-- DroneServiceInterface
|   +-- UserService
|   +-- UserServiceInterface
+-- util
    +-- Hashing
```

Código 5.3: Ruta de archivos del *backend*

[4]El patrón MVC es un patrón que separa la arquitectura en el **modelo**, que representa los datos que se manejan en la lógica (datos peticiones y respuestas, datos obtenidos de sistemas externos, etc), la **vista**, que convierte las respuestas al formato de presentación deseado (HTML, JSON, XML, PDF, CSV, etc.) y el **controlador**, que acepta las peticiones, utiliza el modelo para procesar los datos, que acaba pasando a la vista para generar las respuestas.

Controladores

El controlador es el componente encargado de recibir y gestionar las peticiones HTTP entrantes. Actúa como punto de entrada de la API REST, y está vinculado directamente con los servicios que contienen la lógica de negocio.

En concreto, el controlador define los *endpoints* necesarios para las rutas del usuario (en la Tabla 5.1) y para las rutas de los drones (en la Tabla 5.2).

Cada uno de estos *endpoints* se implementa como un método dentro de una clase anotada con `@RestController`. Las rutas se definen mediante anotaciones como `@GetMapping`, `@PostMapping` o `@RequestMapping`.

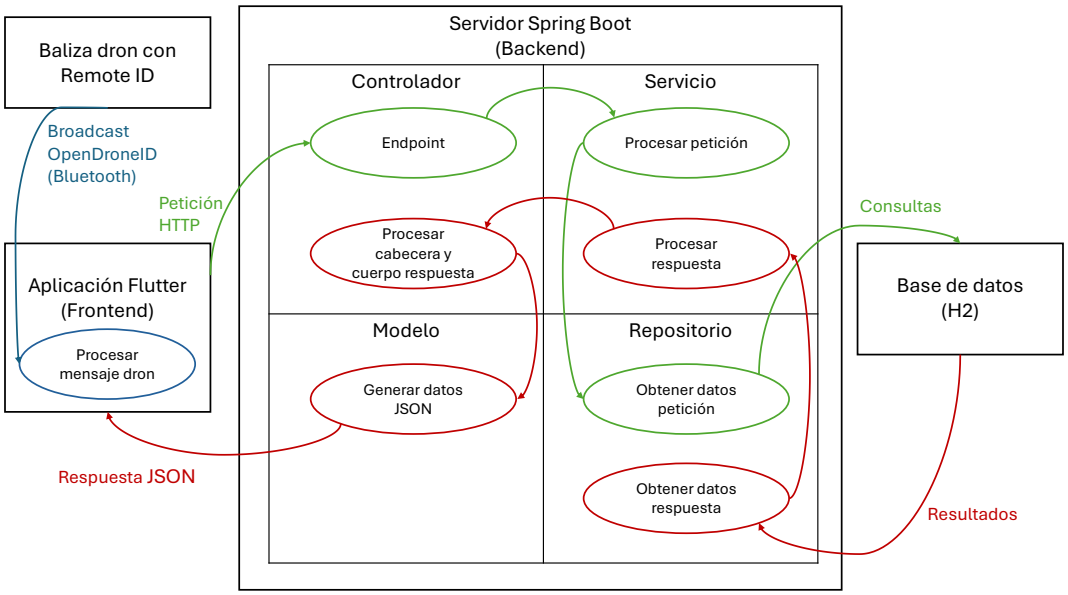

Figura 5.5: Diagrama de caja negra del *backend*.

Servicios

La capa de servicios es la encargada de contener la lógica de negocio de la aplicación. Actúa como intermediaria entre los controladores (que manejan las peticiones HTTP) y los repositorios (que interactúan con la base de datos).

Cada servicio está representado por una clase anotada con `@Service`, que es gestionada automáticamente por el contenedor de Spring. En el *backend* hay uno dedicado a las peticiones de la API de drones y otro dedicado a la API de usuario.

Entidades

Representan las tablas que se almacenan en la base de datos y se modelan como clases Java anotadas con `@Entity`. Como equivalen a las tablas, hay 3 entidades: AppUser, Drone y Token. Cada instancia de una entidad corresponde a una fila de una tabla, y cada atributo representa una columna.

El campo `id` funciona como clave primaria (*primary key*) y se genera automáticamente, y el campo `appUser` funciona como una clave foránea (*foreign key*).

Esta estructura facilita la integración entre el backend y la base de datos relacional.

Repositorios

La capa de repositorios es la encargada de realizar la comunicación directa con la base de datos. En este proyecto se ha utilizado Spring Data JPA, un módulo que permite definir interfaces de acceso a datos sin necesidad de escribir consultas SQL manuales.

Cada entidad del sistema (User, Drone y Token) tiene asociada una interfaz que extiende `JpaRepository`, proporcionando de forma automática métodos como `save()`, `findAll()`, `deleteById()` o `findById()`.

Es posible extender estas funcionalidades con anotaciones como `@Query` para definir consultas personalizadas.

Modelo

Representa los datos que intercambia el *backend* con el exterior a través del controlador. También se han incluido las enumeraciones necesarias para las tablas (entidades) en el modelo.

El uso de modelos permite controlar qué campos son visibles o modificables desde la API y adaptar la salida para que sea más clara o relevante para el cliente.

```
public record RegisterRequest(
    @NotBlank
    String name,
    @NotBlank @Email
    String email,
    @NotNull
    Role role,
    // Patron: al menos una mayuscula, una minuscula, y un
    // numero, y de longitud mas de 7
```

```
    @NotBlank @Pattern(regexp = "^(?=.*[0-9])(?=.*[A-Z])(?=.*[a-z]).{8,}
    $")
    String password
) {}
```

Código 5.4: Ejemplo de Modelo

Útil

Al igual que en el *frontend*, contiene funciones y utilidades auxiliares. En el *backend* del sistema solo se encuentra la clase `Hashing`, que es la responsable del *hashing* de la contraseña.

5.2.6. Diagrama de casos de uso

Un diagrama de casos de uso es un diagrama utilizado en UML(Unified Modeling Language) [26]. UML es utilizado para la estandarización de herramientas de modelación de sistemas como objetos de la vida real. El diagrama de casos de uso [2] muestra cómo un sistema interactúa con entidades externas. El diagrama realizado es la Figura 5.6.

En este diagrama se puede observar que el usuario registrado puede ver el mapa con los drones disponibles por la baliza Bluetooth, además de ver los drones de la base de datos y hacer gestiones básicas de su cuenta.

Lo más interesante de este diagrama de casos de uso es la posibilidad del usuario no registrado de subir los drones que alcance su antena Bluetooth, pero la imposibilidad para recibir los drones de la base de datos. Esto se ha realizado para que cualquier usuario pueda participar en el sistema colaborativo, pero que no sean capaces de aprovechar todos los beneficios del sistema si no están registrados.

5.3. Integración del sistema

Como se indica en el diseño de la comunicación cliente-servidor, la integración del sistema se ha llevado a cabo siguiendo una arquitectura cliente-servidor, donde los distintos componentes desarrollados interactúan de forma coordinada para ofrecer una experiencia funcional y coherente al usuario final. A continuación, se detallan los principales aspectos de esta integración:

- **Integración entre la aplicación móvil y el *backend***: La aplicación Flutter se comunica con el servidor *backend* a través de peticiones HTTP a la API REST expuesta por el sistema.

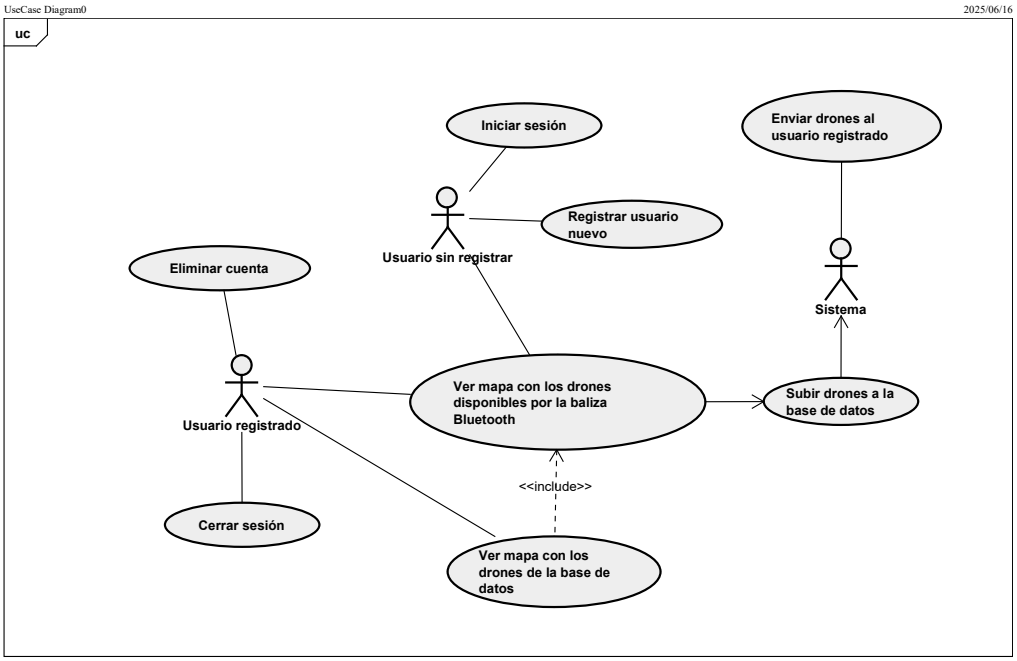

Figura 5.6: Diagrama de casos de uso

- **Integración de la detección BLE y la interfaz**: La aplicación móvil implementa un servicio de escaneo BLE que detecta en tiempo real las tramas emitidas por balizas compatibles con el estándar *Open Drone ID*. La información capturada se interpreta y visualiza automáticamente en la interfaz gráfica, utilizando componentes dinámicos actualizados mediante *ValueNotifier* para reflejar los cambios en el estado de detección.

- **Interacción con la base de datos**: El *backend* desarrollado en Spring Boot emplea Spring Data JPA para mapear entidades Java a una base de datos relacional H2. Cuando un cliente envía datos de detección de drones, el servidor valida, procesa y almacena la información en tablas relacionales, manteniendo la integridad referencial y evitando duplicados mediante restricciones y lógica de servicio.

- **Persistencia de información**: A la vez almacenados, los datos sobre drones detectados en otros envíos se incluyen en la respuesta de la petición para garantizar una respuesta rápida con una sola petición.

La integración ha sido progresiva y modular, permitiendo validar cada componente por separado antes de establecer la conexión entre ellos.

5.3.1. Comunicación entre componentes

El *frontend* y el *backend* se comunican mediante una API REST, utilizando peticiones HTTP (GET y POST).

5.3.2. Verificación de la implantación

Para comprobar que la instalación se ha realizado correctamente:

- Abrir la aplicación en el emulador o dispositivo físico.
- Simular o enviar una ubicación desde el *frontend*.
- Comprobar en la consola del *backend* o en Postman que los datos se han recibido correctamente (Figura B.1.
- Acceder a la consola H2 y verificar que la tabla contiene las ubicaciones enviadas.

5.3.3. Integración de la baliza Open Drone ID

Se ha integrado una baliza que implementa el protocolo Open Drone ID. Esta baliza transmite de forma periódica información relevante sobre el dron, como su identificador único, posición GPS, altitud, velocidad y dirección.

La integración se ha realizado principalmente en el módulo del *frontend*, que se encarga de recibir, procesar y almacenar los datos emitidos por la baliza.

La comunicación entre la baliza y el sistema se realiza utilizando una conexión Bluetooth Low Energy (BLE).

5.3.4. Gestión de localizaciones de los drones

Una de las funcionalidades clave del sistema desarrollado es la capacidad para gestionar las ubicaciones de los drones detectados por la aplicación móvil y almacenarlas en el *backend*. Esta gestión incluye tanto la captura y transmisión de datos de localización como su persistencia y recuperación mediante peticiones HTTP.

Envío de localizaciones desde el cliente

Desde el lado del cliente, como ya se explicó en la sección 5.2.3, se envían estas localizaciones. Cada vez que se detecta un dron que emite su posición mediante una baliza compatible con el estándar Open Drone ID, se construye un modelo con los datos relevantes de este. Esta información se envía al servidor cada segundo utilizando una petición HTTP POST codificada en formato JSON.

Si ningún dron fue localizado por el *frontend*, directamente se solicitarán todos los drones con una petición GET (todas estas peticiones sólo recibirán los drones si el usuario tiene una *cookie* de sesión)

```
Timer.periodic(Duration(seconds: 1), (_) {
  enviarADatabase(dispositivosNotifier.value);
});


Future<void> enviarADatabase(Map<MessageDroneBasicId,
    MessageDroneLocation> dispositivos) async {

if (dispositivos.isEmpty) {
  final List<dynamic> responseJson = await apiServiceDrone.getAllDrones
    ();
  final Map<MessageDroneBasicId, MessageDroneLocation> onlineMap =
    jsonListToDroneMap(responseJson);
  // Actualiza el ValueNotifier de drones online
  onlineDrones.value = onlineMap;
  return;
}
// Prepara la lista de JSONs
final List<dynamic> dronesList = dispositivos.entries.map((entry) {
  final droneMap = Map<dynamic, dynamic>.from(entry.value.toJson());
  droneMap["remoteId"] = entry.key.remoteId;
  droneMap["type"] = entry.key.type.name;
  return droneMap;
}).toList();

// Envia y recibe la lista de drones online
final List<dynamic> responseJson = await apiServiceDrone.uploadDrones(
    dronesList);
final Map<MessageDroneBasicId, MessageDroneLocation> onlineMap =
    jsonListToDroneMap(responseJson);
// Actualiza el ValueNotifier de drones online
```

```
onlineDrones.value = onlineMap;
```

Código 5.5: `Timer` para enviar la petición cada segundo.

Como se puede observar en el Código 5.5, la lista de los drones que se encuentran en la base de datos se recibe al enviar los datos. Esto se hace para poder pedir al servidor que no envíe los drones que el propio dispositivo ha mandado, como respuesta de la petición POST.

Recepción y persistencia en el backend

En el servidor, el controlador correspondiente recibe la petición POST a través de un *endpoint* definido en el controlador de localizaciones. Utilizando Spring Boot y Spring Data JPA, la información se transforma en una entidad persistente que se almacena en la base de datos H2. Como mencionado anteriormente, también devolverá todos los drones encontrados por otros usuarios en otras peticiones.

```
@PostMapping("/api/drone")
@ResponseStatus(HttpStatus.OK)
public List<DroneResponse> uploadDrones(@CookieValue(value = "session",
    required = false) String session, @Valid @RequestBody List<
    DroneRequest> droneRequests) {

     AppUser appUser = null;
     if (session != null){
         appUser = userService.authentication(session);
         if (appUser == null) throw new ResponseStatusException(
    HttpStatus.UNAUTHORIZED);
     }
     // el user puede ser null, no necesita estar registrado
     try {
         List<DroneResponse> drones;
         drones = droneService.uploadDrones(droneRequests, appUser);
         if (appUser != null) {
             if (!drones.isEmpty()) return drones;
             else throw new ResponseStatusException(HttpStatus.NO_CONTENT
    );
         }
         else throw new ResponseStatusException(HttpStatus.
    NON_AUTHORITATIVE_INFORMATION);
     } catch (DataIntegrityViolationException e) {
         throw new ResponseStatusException(HttpStatus.CONFLICT, e.
    getMessage(), e);
     }
}
```

```
@GetMapping("/api/drone/all")
@ResponseStatus(HttpStatus.OK)
public List<DroneResponse> getAllDrones(@CookieValue(value = "session",
    required = true ) String session) {
     AppUser appUser = userService.authentication(session);
     if (appUser == null) throw new ResponseStatusException(HttpStatus.
    UNAUTHORIZED);
     try {
         return droneService.getAllDrones();
     } catch (DataIntegrityViolationException e) {
         throw new ResponseStatusException(HttpStatus.CONFLICT, e.
    getMessage(), e);
     }
}
```

Código 5.6: Endpoints del backend

5.3.5. Gestión del usuario y de la sesión

Esta gestión abarca el registro, la autenticación mediante sesiones basadas en *cookies*, la recuperación de datos del usuario autenticado y la posibilidad de cerrar la sesión. A continuación, se describe cómo se ha implementado este flujo tanto en el *backend* como en el cliente.

Registro e inicio de sesión de usuario

El proceso de registro permite crear nuevas cuentas mediante una petición POST al servidor, enviando los datos básicos (nombre de usuario, contraseña y correo electrónico). En el *backend*, estos datos se validan, se codifica la contraseña y se almacena el nuevo usuario en la base de datos.

El proceso de registro permite acceder a la aplicación con la *cookie* de sesión del usuario, que luego podrá usarse para obtener los drones de la base de datos y para ver la información del usuario.

Gestión de la *cookie* de sesión en el cliente Flutter

En el cliente Flutter, tras el inicio de sesión, se intercepta la *cookie* de sesión y se almacena para ser reutilizada en las siguientes peticiones HTTP. Esto puede hacerse utilizando la librería `http` junto con `shared_preferences` para almacenar la *cookie* de manera local.

```
// Guardar el sessionCookie
Future<void> saveSessionCookie(String sessionCookie) async {
  final prefs = await SharedPreferences.getInstance();
  await prefs.setString('sessionCookie', sessionCookie);
}

// Leer el sessionCookie
Future<String?> loadSessionCookie() async {
  final prefs = await SharedPreferences.getInstance();
  sessionCookie = prefs.getString('sessionCookie');
  return sessionCookie;
}

// Borrar el sessionCookie
Future<void> clearSessionCookie() async {
  final prefs = await SharedPreferences.getInstance();
  await prefs.remove('sessionCookie');
}
```

Código 5.7: Funciones para el almacenamiento de cookies de sesión localmente.

En cada petición, la *cookie* se incluye en la cabecera.

Cierre de sesión

El usuario puede cerrar sesión mediante una petición que invalida la sesión del lado del servidor. Esto elimina la *cookie* de sesión del *backend* y obliga a un nuevo inicio de sesión.

```
public void logout(String tokenId) {
    Optional<Token> optToken = tokenRepository.findById(tokenId);
    if (optToken.isEmpty()){
        return;
    }
    Token token = optToken.get();
    tokenRepository.delete(token);
    return;
}
```

Código 5.8: Servicio del cierre de sesión en el backend

5.3.6. Cifrado de contraseñas mediante hashing

Para proteger las credenciales de los usuarios registrados en el sistema, se ha implementado un mecanismo de cifrado basado en *hashing* con *salted hashing*. Esta técnica garantiza que las contraseñas almacenadas en la base de datos no puedan ser recuperadas en texto plano.

Algoritmo utilizado

El sistema utiliza el algoritmo `PBKDF2WithHmacSHA1` con una sal generada aleatoriamente para cada contraseña. La combinación de contraseña y sal se procesa mediante una función derivadora de clave (PBKDF2), con 65536 iteraciones y una clave final de 128 bits. La salida se codifica en *Base64* y se almacena junto con la sal, también codificada.

Implementación en Java

La lógica de *hashing* se encuentra encapsulada en la clase `Hashing`, ubicada en el paquete `util`. Esta clase proporciona dos métodos principales: `hash()`, para generar una cadena cifrada a partir de una contraseña, y `compare()`, para validar una contraseña introducida por el usuario frente a la almacenada.

Uso en el proceso de registro y login

Durante el registro de un nuevo usuario, la contraseña proporcionada se hashea mediante el método `hash()` y se almacena en la base de datos junto con el resto de los datos del usuario. En el proceso de autenticación (login), la contraseña introducida se compara con el valor almacenado usando `compare()`. Si los hashes coinciden, el acceso es concedido.

5.3.7. Integración del escáner Bluetooth y simulación mediante proxy

El proceso de integración del escáner Bluetooth se basa en la escucha activa de mensajes emitidos por balizas compatibles con el estándar Open Drone ID. Esta funcionalidad permite detectar y procesar información de identificación y localización transmitida por drones cercanos mediante Bluetooth LE gracias a la librería Flutter Open Drone ID [8].

Procesamiento de mensajes Open Drone ID

La clase encargada de gestionar los mensajes recibidos `FlutterOdidBluetoothScanner` inicia un escaneo mediante la función `startScan()` de la librería `FlutterOpenDroneId`. Cada mensaje recibido se procesa para extraer tanto la dirección MAC como los datos contenidos en los mensajes de tipo `Basic ID` y `Location` (los demás mensajes que envía la baliza del dron no son necesarios para el funcionamiento del sistema realizado).

Ambos mensajes contienen la dirección MAC del emisor, lo que permite asociar información entre tipos de mensajes distintos y realizar un seguimiento temporal del dispositivo. En concreto, se mantienen tres estructuras internas:

- `macLastSeen`: almacena el instante exacto en que se recibió por última vez un mensaje desde una MAC determinada.
- `macToDroneId`: asocia cada dirección MAC con su identificador remoto (`remote ID`).

A partir de estos datos se construye el mapa `dispositivosAnalizados`, que almacena información agregada por dron detectado (Basic ID y Location). Esta estructura es gestionada mediante un `ValueNotifier`[5], de forma que cualquier cambio en el conjunto de drones visibles se refleja automáticamente en la interfaz de usuario.

```
FlutterOpenDroneId.startScan(DriSourceType.Bluetooth);

_odidSubscription = FlutterOpenDroneId.bluetoothMessages.listen((
    container) {
  final mac = container.metadata.macAddress;
  macLastSeen[mac] = DateTime.now();

  final BasicIDMessage? messageIdMessage = container.basicIdMessages?.
    values.first;
  final LocationMessage? messageLocationMessage = container.
    locationMessage;

  if (messageIdMessage is BasicIDMessage) {
    macToDroneId[mac] = getIdValue(messageIdMessage.uasID, mac);
```

[5] `ValueNotifier` es una clase que permite almacenar un valor reactivo y notificar automáticamente a los *widgets* que lo escuchan cuando dicho valor cambia. Este patrón ha sido empleado para controlar elementos dinámicos en la interfaz, como la actualización de los drones de manera local.

```
  }
  if (messageLocationMessage is LocationMessage &&
   messageLocationMessage.location != null) {
    final droneId = macToDroneId[mac];
    if (droneId != null) {
      final newMap = Map<MessageDroneBasicId, MessageDroneLocation>.from
    (dispositivosAnalizadosNotifier.value);
      newMap[droneId] = MessageDroneLocation(
        longitude: messageLocationMessage.location?.longitude,
        latitude: messageLocationMessage.location?.latitude,
        ...
        height: messageLocationMessage.height,
      );
      dispositivosAnalizadosNotifier.value = newMap;
    }
  }
});
```

Código 5.9: Identificar mensajes de Open Drone ID

Limpieza de MACs inactivas

Para evitar que drones inactivos o fuera de alcance permanezcan visibles indefinidamente, se ejecuta periódicamente una limpieza de direcciones MAC no vistas recientemente. Si ha pasado un tiempo mayor a un umbral predefinido desde la última recepción, se eliminan sus datos del mapa `dispositivosAnalizados`.

```
FlutterOdidBluetoothScanner() {
  cleanupTimer = Timer.periodic(Duration(seconds: TIME_CLEANUP), (_) {
    limpiarMacsInactivas();
  });
}

void limpiarMacsInactivas() {
  final now = DateTime.now();
  final toRemove = <String>[];
  macLastSeen.forEach((mac, lastSeen) {
    if (now.difference(lastSeen).inSeconds > MAX_SECONDS) {
      toRemove.add(mac);
    }
  });
  for (final mac in toRemove) {
    final droneId = macToDroneId[mac];
    macToDroneId.remove(mac);
```

```
    macLastSeen.remove(mac);
    if (droneId != null) {
      dispositivosAnalizadosNotifier.value.remove(droneId);
    }
  }
}
```

Código 5.10: Función para limpiar las MACs inactivas

Simulación mediante *proxy* Bluetooth

Para facilitar las pruebas sin necesidad de disponer de un dron físico, se ha desarrollado un módulo *proxy* que simula el comportamiento de una baliza Bluetooth. Este *proxy* envía cada segundo dos tipos de mensaje:

- Un mensaje de tipo `Basic ID`, que contiene un falso remote ID.
- Un mensaje de tipo `Location`, con coordenadas ligeramente modificadas en cada iteración para simular movimiento.

Ambos mensajes se emiten usando la misma MAC falsa, lo que permite que el sistema los relacione como si fueran de un mismo dron. A los 10 segundos de iniciada la simulación, se detiene el envío de mensajes de localización, simulando así que el dron ha salido del alcance del escáner. Este comportamiento permite verificar que el sistema reacciona correctamente ante la desaparición de dispositivos, eliminándolos del mapa tras un cierto tiempo de inactividad.

Consideraciones de rendimiento

Dado que la base de datos empleada es H2 en modo local, y el volumen de datos manejado es limitado, no se han implementado estrategias de optimización avanzadas. Sin embargo, el diseño de las entidades y repositorios está preparado para escalar a una base de datos relacional más robusta como PostgreSQL en un entorno de producción.

La implementación del sistema ha permitido materializar las decisiones de diseño. La integración entre los distintos componentes se ha realizado de forma modular y escalable, respetando los principios de separación de responsabilidades y reutilización de código.

5.4. Pruebas realizadas

Durante el desarrollo del sistema se han llevado a cabo diversas pruebas para asegurar la corrección de los componentes implementados. Estas pruebas han abarcado desde validaciones unitarias sobre objetos de datos hasta pruebas *end-to-end* (E2E) sobre el sistema completo. A continuación se describen los distintos tipos de pruebas realizadas:

Pruebas unitarias

Se han desarrollado pruebas unitarias para verificar el comportamiento de clases concretas, como los objetos de transferencia de datos (DTO). Por ejemplo, se ha validado que la clase `RegisterRequest` cumple correctamente con las restricciones impuestas mediante anotaciones de validación, tales como formato de correo electrónico y requisitos mínimos de contraseña.

- Se verifica que una solicitud de registro válida no genera violaciones.
- Se comprueba que correos inválidos o contraseñas débiles producen errores de validación.

Pruebas de integración sobre repositorios

Utilizando la anotación `@DataJpaTest`, se han probado los repositorios para garantizar que las entidades se persisten y recuperan correctamente en la base de datos. Las pruebas incluyen:

- Almacenamiento y recuperación de usuarios, drones y tokens.
- Asociación entre entidades (por ejemplo, un token asociado a un usuario).
- Borrado en cascada: al eliminar un usuario, se eliminan también sus drones y tokens asociados.

Estas pruebas permiten comprobar la configuración correcta de relaciones entre entidades y del contexto de persistencia en Spring Data JPA.

Pruebas de integración sobre controladores

Mediante el uso de `@SpringBootTest` y `@AutoConfigureMockMvc`, se han implementado pruebas sobre los controladores del *backend*, simulando peticiones HTTP para verificar los distintos *endpoints*.

Entre los escenarios cubiertos se incluyen:

- Registro de usuario con datos válidos e inválidos.
- Subida de drones por un usuario autenticado (mediante *cookie*).
- Comprobación del comportamiento del sistema ante accesos no autorizados.
- Eliminación y recuperación de drones a través de las rutas expuestas.

Estas pruebas aseguran que la capa de presentación se comporta como se espera ante distintos flujos de usuario.

Pruebas *end-to-end* (E2E)

Finalmente, se han desarrollado pruebas de extremo a extremo utilizando `TestRestTemplate`, ejecutando el servidor en un puerto real con configuración completa. Estas pruebas permiten validar el funcionamiento del sistema como un todo, incluyendo:

- Registro e inicio de sesión de un usuario completo, incluyendo la gestión de *cookies* de sesión.
- Subida de drones autenticado y no autenticado.
- Comprobación del flujo completo desde el inicio de sesión hasta el envío de datos.

Estas pruebas reflejan el comportamiento del sistema en condiciones reales, tal como lo experimentaría un cliente (por ejemplo, la app Flutter).

Capítulo 6

Análisis de Resultados

6.1. Cumplimiento de objetivos

A lo largo del desarrollo del proyecto se han alcanzado los objetivos establecidos en la fase inicial, tanto a nivel funcional como técnico. A continuación, se analiza el grado de cumplimiento de cada uno de ellos:

- **Objetivo general.** Se ha implementado un sistema completo que permite detectar drones mediante balizas Open Drone ID, visualizar sus datos en una aplicación móvil desarrollada con Flutter y registrar dicha información en un *backend* centralizado basado en Spring Boot, expuesto mediante una API REST y respaldado por una base de datos relacional (H2).
- **Objetivo específico 1.** Se ha desarrollado una interfaz de usuario funcional que permite el registro e inicio de sesión de usuarios, así como la detección y escaneo de tramas BLE emitidas por balizas Open Drone ID. Esta funcionalidad ha sido validada mediante pruebas de recepción de señales en dispositivos reales.
- **Objetivo específico 2.** El sistema permite almacenar correctamente las localizaciones recibidas en una base de datos relacional, y permite devolver todos los drones almacenados salvo los drones que se ha enviado en la petición.
- **Objetivo específico 3.** La base de datos ha sido modelada e implementada con un esquema estructurado que recoge la posición, el identificador y la última vez que se registró el dron de cada dron. El uso de Spring Data JPA garantiza la integridad de los datos y facilita su consulta.
- **Objetivo específico 4.** La arquitectura del sistema sigue un modelo por capas tanto en el cliente como en el servidor. En el *backend*, se han definido

capas diferenciadas para controladores, servicios, repositorios, entidades y modelos, mientras que en el *frontend* se ha mantenido una separación clara entre páginas y servicios. Esta organización favorece el mantenimiento del código y cumple con los principios de cohesión y bajo acoplamiento.

En resumen, los objetivos planteados al inicio del proyecto se han cumplido satisfactoriamente, dando lugar a una solución funcional, modular y extensible, que puede servir como base para futuras ampliaciones del sistema.

6.2. Interfaz de usuario

La aplicación móvil desarrollada en Flutter presenta una interfaz sencilla y funcional que permite al usuario interactuar de forma intuitiva con el sistema. A continuación, se muestran algunas de las pantallas principales de la aplicación que reflejan el cumplimiento de los objetivos definidos.

La Figura 6.1 y la Figura 6.2 muestran las pantallas de autenticación. Se puede cambiar entre la pantalla de inicio de sesión y la pantalla de registro, además de poder continuar en local sin registrar al usuario.

Figura 6.1: Página de inicio de sesión.

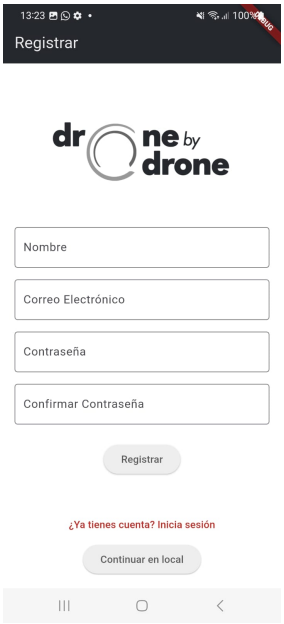

Figura 6.2: Página de registro.

En la Figura 6.4 y la Figura 6.3 se observan dos funcionalidades clave de la app: la visualización de los drones detectados mediante balizas Open Drone ID y la consulta de información detallada (posición, identificador, etc.). Esta información se obtiene en tiempo real desde las tramas BLE capturadas por el dispositivo. Estas ubicaciones se marcan con marcadores rojos. Además, se pueden ver los drones encontrados en la base de datos con los marcadores verdes, para que no se confundan con los drones locales. Si el usuario presiona la ubicación de un dron, este accederá a una pantalla con los detalles del mismo (Figura 6.5). También se muestra la ubicación del usuario con un marcador azul.

Además, tienes la opción de ir a la página de información del usuario. En la Figura 6.6 se puede ver esta página, en la que se puede eliminar el usuario y cerrar sesión. Si el dispositivo no está registrado, llevará a la página de autenticación (como se puede ver en la Figura 5.3).

En conjunto, la interfaz desarrollada cumple con los principios de usabilidad, modularidad y claridad, permitiendo una experiencia fluida en dispositivos móviles con recursos limitados.

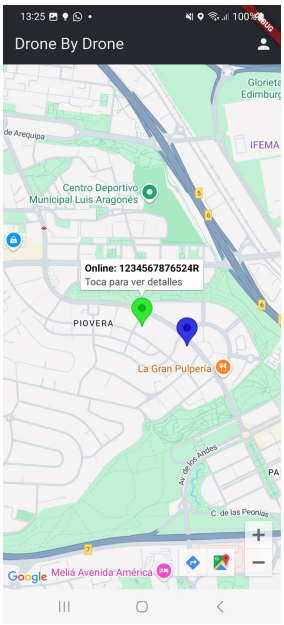

Figura 6.3: Mapa con dron de la base de datos.

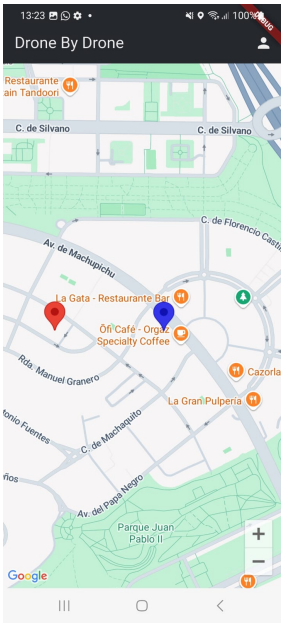

Figura 6.4: Mapa con dron detectado.

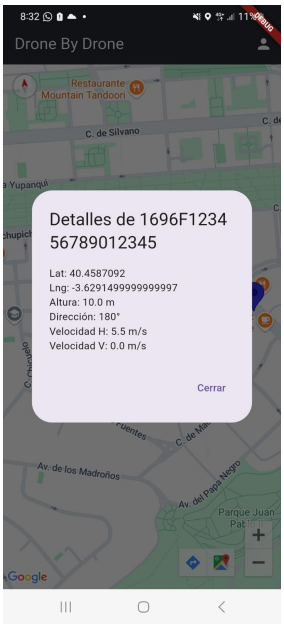

Figura 6.5: Información detallada del dron.

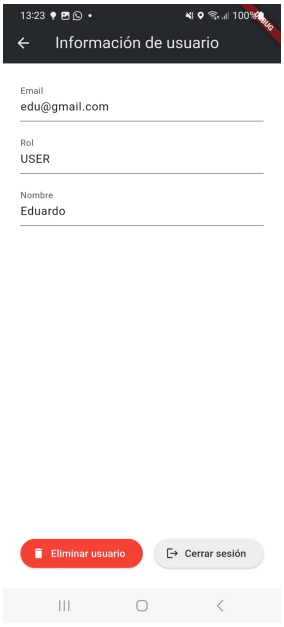

Figura 6.6: Página de información del usuario.

Capítulo 7

Conclusiones y Trabajos Futuros

El desarrollo de este Trabajo de Fin de Grado ha permitido diseñar e implementar un sistema colaborativo de detección de drones equipados con balizas de identificación remota compatibles con el estándar Open Drone ID. A lo largo del proceso, se han aplicado conocimientos aprendidos en el Grado de Tecnologías de la Telecomunicación, abarcando tanto el desarrollo móvil como el diseño de arquitecturas cliente-servidor, la gestión de comunicaciones Bluetooth Low Energy (BLE) y el modelado de bases de datos relacionales mediante Spring Boot.

Desde el punto de vista funcional, se ha logrado implementar una aplicación móvil desarrollada con Flutter, capaz de escanear y decodificar tramas BLE emitidas por las balizas de los drones, mostrando en tiempo real la información recibida. Esta información incluye datos clave como la posición, la hora de recepción del mensaje y el identificador único del dron, todo ello presentado mediante una interfaz gráfica intuitiva. Además, la aplicación permite a los usuarios registrarse, iniciar sesión y esta comparte automáticamente la información detectada con el resto del sistema, fomentando así una red colaborativa de detección.

En paralelo, se ha desarrollado un *backend* robusto utilizando el *framework* Spring Boot, que expone una API REST para recibir y distribuir los datos de los drones detectados. La persistencia se ha resuelto mediante una base de datos relacional H2, lo que ha facilitado el desarrollo local y las pruebas sin necesidad de recursos en la nube. La arquitectura en capas del servidor, dividida en controladores, servicios, repositorios y modelo, ha permitido un desarrollo limpio, modular y fácilmente extensible, cumpliendo con principios aprendidos en el grado de ingeniería de software como la separación de responsabilidades, la cohesión interna y el bajo acoplamiento.

Los objetivos definidos al inicio del proyecto se han alcanzado en su totalidad.

Se ha cumplido tanto el objetivo general como los objetivos específicos, que abarcaban desde la implementación de la interfaz de usuario hasta la persistencia de datos y la estructura arquitectónica del sistema.

Durante el desarrollo, se han superado diversas dificultades técnicas, especialmente en la integración con dispositivos BLE y en la interpretación del estándar Open Drone ID. Esto ha sido complicado por la falta de recursos encontrados acerca de este estándar, y el descubrimiento de la librería `flutter_opendroneid` [8] ha sido clave para la elaboración de este trabajo.

Además del desarrollo técnico, este proyecto ha servido como experiencia integral de gestión de software, incluyendo fases de análisis, diseño, implementación, pruebas y validación. Se ha llevado a cabo una planificación temporal mediante diagrama de Gantt, siguiendo buenas prácticas del ciclo de vida de un proyecto de software.

Como resultado, se ha obtenido un sistema funcional, modular, extensible y adaptable, que podría evolucionar hacia una solución más completa en el ámbito de la monitorización de espacio aéreo no tripulado.

Algunas líneas de trabajo futuras podrían incluir el almacenamiento persistente en bases de datos PostgreSQL o MySQL, las cuales están más orientadas a producción y aplicaciones a gran escala. Otra funcionalidad que se podría añadir es agregar contenido exclusivo de administrador como la posibilidad de que los drones subidos por ellos sólo puedan acceder ellos mismos u otros usuarios elegidos por ellos en un futuro. También podría contemplarse para un futuro la incorporación de filtros por zonas e información para el buen uso de los drones. Además, algo que sería clave para que la aplicación saliese al mercado sería el uso de un servidor en la nube que tenga el *backend* y la base de datos (como por ejemplo AWS) y el uso de pruebas de robustez y latencia para adaptarse a un gran número de usuarios. En el caso de que se fuera conveniente, se podría incluir también la detección de balizas por Wi-Fi, y se podría plantear si existen un gran número de drones en la base de datos que el dispositivo solamente reciba los que estén a una distancia específica de ellos.

En conclusión, este proyecto ha permitido demostrar la viabilidad técnica de un sistema distribuido de localización de drones basado en tecnologías modernas y estándares abiertos. Me ha permitido unificar contenidos estudiados durante el grado en un trabajo con un aspecto práctico y real, respondiendo a necesidades actuales del mercado y a una aportación a la empresa Drone By Drone.

Bibliografía

[1] Drone by Drone. *Drone by Drone*. https://www.dronebydrone.com/. Accessed: 2025-06-04.

[2] Alistair Cockburn. *Writing Effective Use Cases*. Addison–Wesley Professional, 2000. ISBN: 978-0201702255.

[3] M. M. Bonilla Yoza. "EL USO DE LOS DRONES EN EL ÁMBITO PROFESIONAL: EL USO DE LOS DRONES EN EL ÁMBITO PROFESIONAL". En: *Ciencias. Revista Científica Multidisciplinaria* (2021).

[4] M. Ghasri y M. Maghrebi. "Factors affecting unmanned aerial vehicles' safety: A post-occurrence exploratory data analysis of drones' accidents and incidents in Australia". En: *Safety Science* (2021).

[5] European Union Aviation Safety Agency. *Remote identification will become mandatory for drones across Europe*. 2023.

[6] ASTM International. *ASTM International – Standards Worldwide*. Accedido el 8 de junio de 2025. 2024. URL: https://www.astm.org.

[7] Google LLC. *Flutter*. Accedido el 9 de junio de 2025. 2024. URL: https://flutter.dev.

[8] Dronetag. *flutter-opendroneid*. Repositorio en GitHub, accedido el 9 de junio de 2025. 2024. URL: http://github.com/dronetag/flutter-opendroneid.

[9] Flutter Team. *shared_preferences*. https://pub.dev/packages/shared_preferences. Accedido el 18 de junio de 2025. 2024.

[10] Google LLC. *Dart Programming Language*. Accedido el 8 de junio de 2025. 2024. URL: https://dart.dev.

[11] Federal Aviation Administration. *Federal Aviation Administration (FAA)*. Accedido el 8 de junio de 2025. 2024. URL: https://www.faa.gov.

[12] ASTM International. *Standard Specification for Remote ID and Tracking. ASTM F3411-22a*. Accedido el 8 de junio de 2025. 2022. URL: https://www.astm.org/f3411-22a.html.

[13] Dronetag. *Google Play*. https://play.google.com/store/apps/details?id=cz.dronetag.dronescanner&hl=nl. Accessed: 2025-06-04.

[14] Robin Andrén et al. "Open Drone ID - Remote Identification for Drones". En: *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2022, págs. 1327-1336. DOI: 10.1109/ICUAS54217.2022.9861637.

[15] Spring Team. *Spring Boot*. Accedido el 8 de junio de 2025. 2024. URL: https://spring.io/projects/spring-boot.

[16] Eclipse Foundation. *Jakarta Persistence (JPA)*. Especificación oficial de JPA, accedido el 8 de junio de 2025. 2024. URL: https://jakarta.ee/specifications/persistence.

[17] Spring Team. *Spring Data JPA*. Accedido el 8 de junio de 2025. 2024. URL: https://spring.io/projects/spring-data-jpa.

[18] Thomas Mueller. *H2 Database Engine*. Accedido el 8 de junio de 2025. 2024. URL: https://www.h2database.com.

[19] Roy T. Fielding y Julian F. Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. RFC 7230, Internet Engineering Task Force (IETF). Accedido el 8 de junio de 2025. 2014. URL: https://datatracker.ietf.org/doc/html/rfc7230.

[20] Postman, Inc. *Postman API Platform*. Accedido el 8 de junio de 2025. 2024. URL: https://www.postman.com.

[21] GitHub, Inc. *GitHub*. https://github.com. Último acceso: 30 de junio de 2025. 2024.

[22] K. Belwafi et al. "Unmanned Aerial Vehicles' Remote Identification: A Tutorial and Survey". En: *IEEE Access* (2022), págs. 87577-87601.

[23] E. Vinogradov y S. Pollin. "Reducing safe UAV separation distances with U2U communication and new Remote ID formats". En: *IEEE* (2022).

[24] Commonwealth of Australia. *Remote Identification (Remote ID): Discussion Paper for Public Consultation*. Australia. 2023.

[25] Donetonic. *Metodología Waterfall vs Metodología Agile: ¿Cuál es mejor?* Accedido el 17 de junio de 2025. 2023. URL: https://donetonic.com/es/metodologia-waterfall-vs-metodologia-agile/.

[26] Object Management Group. *OMG Unified Modeling Language (UML), Version 2.5.1*. Accedido el 8 de junio de 2025. 2017. URL: https://www.omg.org/spec/UML/2.5.1/PDF.

[27] Naciones Unidas. *Objetivos de Desarrollo Sostenible (ODS)*. https://www.cepal.org/es/temas/agenda-2030-desarrollo-sostenible/objetivos-desarrollo-sostenible-ods. CEPAL, s.f., Accessed: 2025-06-04.

Apéndice A

Alineación con los Objetivos de Desarrollo Sostenible (ODS)

Figura A.1: Objetivos de Desarrollo Sostenible [27].

Los Objetivos de Desarrollo Sostenible (ODS) son un conjunto de 17 metas globales establecidas por las Naciones Unidas en 2015 como parte de la Agenda 2030. Según las Naciones Unidas, "muestran una mirada integral, indivisible y una colaboración internacional renovada. En conjunto, construyen una visión del futuro que queremos." [27].

Este proyecto se alinea con ciertos objetivos y contribuye a su realización. Los objetivos más directamente relacionados son:

- **ODS 9:** Construir infraestructuras resilientes, promover la industrialización sostenible y fomentar la innovación. Esta aplicación pretende aportar una

tecnología que promueva la utilización de drones en sectores nuevos, fomentando así la industrialización sostenible y el uso de esta tecnología con alto potencial.

- **ODS 11:** Lograr que las ciudades sean más inclusivas, seguras, resilientes y sostenibles. Uno de los objetivos claves del proyecto es contribuir a que las ciudades sean más seguras, evitando posibles accidentes relacionados con UAVs.

Apéndice B

Manual de Instalación

Este manual de Instalación describe cómo se puede instalar y desplegar el sistema. Este se encuentra en un repositorio GitHub, en el URL `https://github.com/igarrastazu/drone_by_drone`.

B.1. Requisitos previos

Para ejecutar el sistema se requiere tener instalados los siguientes componentes:

- **Flutter SDK** (versión 3.6.1 o superior).
- **Java JDK** (versión 17 o superior).
- **Maven**, como herramienta de construcción para el backend.
- **Android Studio** o **Visual Studio Code**, con soporte para Flutter.
- **Emulador de Android** o dispositivo físico (con el modo depuración por USB activado).
- **Postman** (opcional) para probar la API REST del backend. Se puede ver un ejemplo de una petición en Postman en la Figura B.1.

B.2. Implantación del *backend*

El *backend* ha sido desarrollado con Spring Boot y se ejecuta como una aplicación Java.

1. Clonar el repositorio del backend.

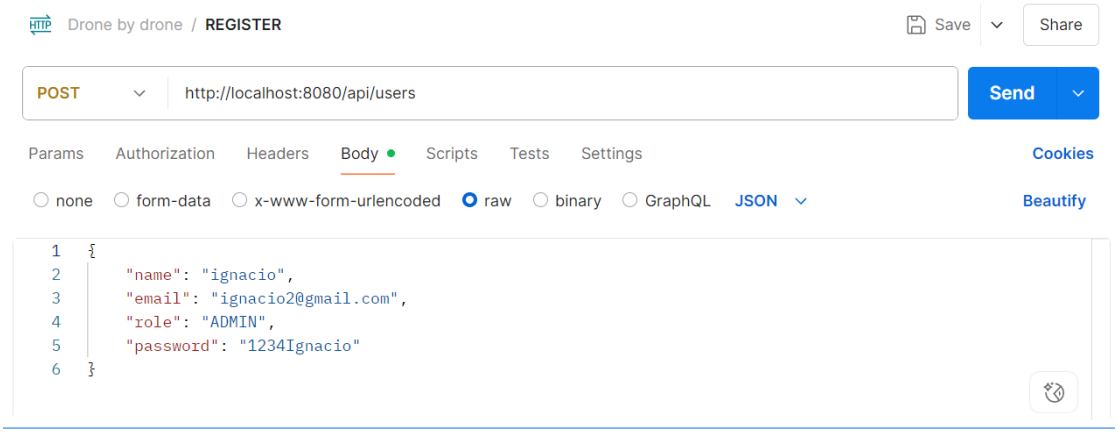

Figura B.1: Captura de pantalla de una petición en Postman.

2. Ejecutar el backend con Maven:

   `mvn spring-boot:run`

3. El *backend* estará disponible en `http://localhost` y, para dispositivos en tu red, en la IP que tenga asignada de la red (se puede mirar en el terminal con el comando `ip-config`) con el puerto `8080`.

4. La consola H2 se puede consultar en `http://localhost:8080/h2-console`, utilizando la URL `jdbc:h2:file:./dbdata` y el usuario `sa` (sin contraseña) como se puede observar en la Figura B.2.

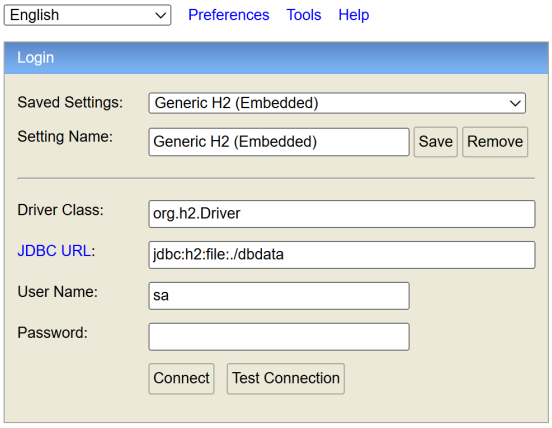

Figura B.2: Acceso a la consola H2

B.3. Implantación del *frontend*

El *frontend* se ha desarrollado en Flutter. Para su ejecución es importante tener un emulador instalado o bien conectar un teléfono Android con el modo desarrollador y la depuración USB activada (opción recomendable, para poder utilizar la geolocalización y el Bluetooth). Habiendo hecho esto, los siguientes pasos a seguir son:

1. Clonar el repositorio de la aplicación móvil.
2. Ejecutar en terminal el comando para obtener las dependencias:

   `flutter pub get`
3. Reemplazar la URL del backend por la IP del servidor en tu red (se puede mirar en el terminal del servidor con ipconfig)
4. Ejecutar la aplicación con:

   `flutter run`

Apéndice C

Manual de Usuario

Este manual de usuario describe el funcionamiento de la aplicación móvil desarrollada para la localización de drones mediante balizas compatibles con el estándar Open Drone ID. Se detallan las principales funcionalidades, el flujo de uso, los requisitos técnicos y algunas recomendaciones básicas para el correcto funcionamiento de la aplicación.

C.1. Requisitos del sistema

- **Sistema operativo:** Android.
- **Conectividad:** Bluetooth activado y permisos concedidos.
- **Permisos requeridos:** Acceso a la ubicación, uso de Bluetooth.
- **Conexión a internet** para sincronización con el servidor.

C.2. Instalación

1. Descargar el archivo APK desde el repositorio del proyecto o mediante el enlace proporcionado.
2. Instalar la aplicación habilitando la instalación desde fuentes desconocidas si es necesario.
3. Asegurarse de conceder todos los permisos solicitados al abrir la aplicación por primera vez.

C.3. Inicio de sesión y registro

- Al abrir la aplicación la primera pantalla que aparece es la de inicio de sesión (Figura C.1). Si el usuario tiene cuenta tendrá que introducir sus credenciales. Si el usuario no tiene cuenta, podrá dar al botón de registrarse o de continuar en local.
- Para registrarse, el usuario debe introducir un nombre, una dirección de correo electrónico válida y una contraseña (Figura **??**).

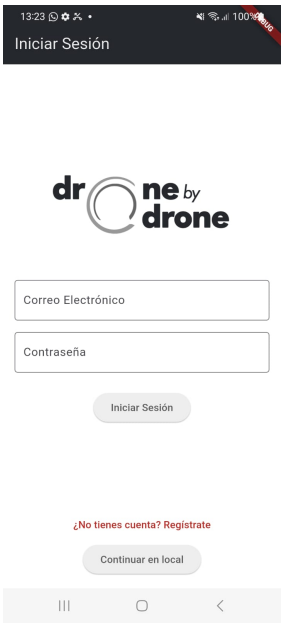

Figura C.1: Pantalla de inicio de sesión.

C.4. Interfaz principal

Una vez autenticado, el usuario accede a la interfaz principal, donde se encuentra el mapa.

- Los **drones detectados** se muestran en el mapa con su identificador con un marcador rojo.
- Los **drones recibidos** se muestran en el mapa con su identificador con un marcador verde.

Figura C.2: Pantalla de registro.

- La **ubicación del usuario** se encuentra en el mapa con un marcador azul.

En ambos casos, si se selecciona el dron en el mapa, se puede obtener información más detallada de este, como su velocidad y dirección (Figura C.4).

C.5. Información del usuario

Se puede acceder a este menú con la esquina superior del mapa en el icono del usuario (Figura C.4). Si se ha iniciado sesión, se mostrarán las siguientes opciones:

- Cierre de sesión desde el menú de perfil.
- Eliminación de cuenta si el usuario lo desea.

Si el usuario no ha iniciado sesión, este icono le llevará a la pantalla de autenticación.

C.6. Recomendaciones

- Mantener el Bluetooth y la ubicación activados durante el uso de la app.

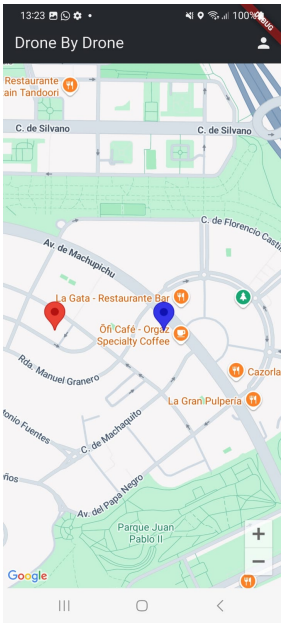

Figura C.3: Mapa con un dron localizado con el dispositivo

- Estar en exteriores o zonas despejadas para una mejor recepción de las balizas.
- Conectarse a una red Wi-Fi o tener buena cobertura de datos móviles para sincronizar correctamente.

C.7. Resolución de problemas

- **No se detectan drones:** Asegurarse de que el Bluetooth está activado y que los permisos están concedidos.
- **Error de autenticación:** Comprobar que el correo y la contraseña son correctos o intentar registrarse nuevamente.
- **Fallo de sincronización:** Revisar la conexión a internet.

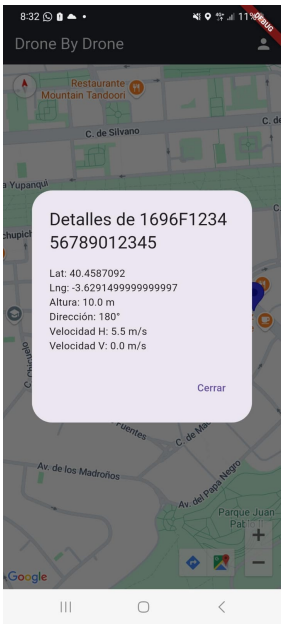

Figura C.4: Información detallada del dron

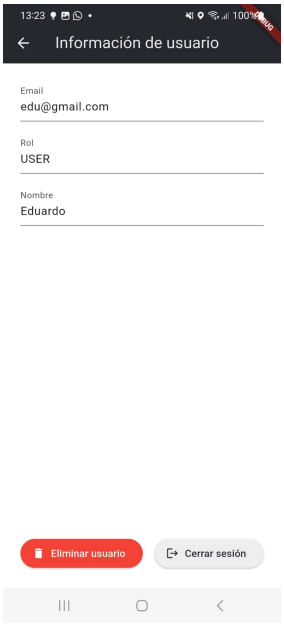

Figura C.5: Información del Usuario.