Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN INGENIERÍA INFORMÁTICA DE GESTIÓN Y SISTEMAS DE INFORMACIÓN

# TRABAJO FIN DE GRADO

# *AGROFIND: APLICACIÓN SERVERLESS PARA TRAZABILIDAD AGRÍCOLA MEDIANTE BLOCKCHAIN*

AGROFIND


**Estudiante**: Mena Ruiz, Adrian.

**Director:** López Novoa, Unai.

**Curso:** 2024-2025

**Fecha:** Bilbao, 23 de junio de 2025.

# Resumen

El avance de la tecnología ha provocado que sectores como la agricultura queden, en ocasiones, rezagados frente a otras industrias más digitalizadas. No obstante, el sector primario sigue siendo un pilar esencial, y su modernización resulta clave. En este contexto, la tecnología *blockchain* se presenta como una herramienta con gran potencial para mejorar la trazabilidad y la transparencia en la cadena de suministro de productos agrícolas.

Aunque el término de blockchain se asocia habitualmente al ámbito financiero, sus aplicaciones se han expandido a numerosos sectores. Este proyecto propone aprovechar su capacidad para registrar información de forma segura, inmutable y verificable mediante *smart contracts*, aplicándolo al seguimiento de la cosecha de los cultivos.

**AgroFind** consiste en una aplicación que permite a los agricultores documentar con precisión cada etapa del proceso agrícola, desde la siembra hasta la cosecha. Los consumidores, por su parte, podrán consultar dicha información mediante códigos QR. Para ello, se emplea un enfoque *serverless*, que elimina la necesidad de servidores propios, y se integra el sistema SIGPAC, lo que permite identificar oficialmente las parcelas agrícolas de forma sencilla y precisa.

La aplicación ha sido diseñada con un enfoque centrado en la accesibilidad y facilidad de uso, especialmente pensada para usuarios de entornos rurales con poca experiencia tecnológica. El desarrollo se realiza en Flutter, lo que permite una solución multiplataforma compatible con diferentes sistemas operativos.

**Palabras clave**: Agricultura, trazabilidad, blockchain, serverless, SIGPAC, Flutter, aplicación multiplataforma.

# Laburpena

Teknologiaren aurrerapenak nekazaritza bezalako sektoreak digitalizatuago dauden beste industria batzuen aurretik geratzea eragin du. Hala ere, lehen sektorea oso garrantzitsua da oraindik, eta bere modernizazioa ezinbestekoa da. Testuinguru honetan, *blockchain* teknologia nekazaritzako produktuen hornidura-katean trazabilitatea eta gardentasuna hobetzeko tresna gisa bezala aurkezten da.

Blockchain-a finantzen esparruarekin lotu ohi den arren, bere aplikazioak sektore askotara hedatu dira. Proiektu honek informazioa modu seguruan, aldaezinean eta egiaztagarrian *smart contracts*-en bidez erregistratzeko duen gaitasuna aprobetxatzea proposatzen du, nekazal laboreen jarraipenerako.

**AgroFind** baserritarrei nekazaritza-prozesuaren pausu bakoitza zehaztasunez dokumentatzeko aukera ematen dien aplikazio bat da, ereintzatik uztaraino. Kontsumitzaileek bestalde, QR kodeen bidez kontsultatu ahal izango dute informazio hori. Horretarako, *serverless* ikuspegia erabiltzen da, zerbitzari propioen beharra ezabatzen duena, eta SIGPAC sistema integratzen da, nekazaritzako lursailak modu erraz eta zehatzean ofizialki identifikatzeko aukera ematen duena.

Aplikazioa erabiltzeko erraza eta intuitiboa izatearen ikuspegiarekin diseinatu da, bereziki trebezia teknologiko gutxi dituzten landa inguruneetako erabiltzaileentzat. Garapena Flutterren egiten da, eta horrek hainbat sistemekin bateragarria den plataforma anitzeko soluzioa ahalbidetzen du.

**Gako-hitzak:** Nekazaritza, trazabilitatea, blockchain, serverless, SIGPAC, Flutter, plataforma-anitzeko aplikazioa.

# Abstract

The advancement of technology has led to sectors like agriculture to get behind more than other digitized industries. However, the primary sector remains an essential pillar, and its modernization is vital. In this context, *blockchain* technology emerges as a tool with great potential to improve traceability and transparency in the agricultural supply chain.

Although the blockchain is commonly associated with the financial sector, its applications have expanded to numerous industries. This project proposes leveraging its ability to securely, immutably, and verifiably record information through *smart contracts*, applying it to track the harvest of the crops.

**AgroFind** is an application that allows farmers to accurately document each stage of the agricultural process, from sowing to the harvest. Consumers, in turn, can access this information via QR codes. To achieve this, a serverless approach is used, eliminating the need for proprietary servers, and the SIGPAC system is integrated, enabling the official and precise identification of agricultural plots.

The application has been designed with a focus on accessibility and ease of use, especially tailored for rural environments with limited technological expertise. The development is carried out in Flutter, allowing for a cross-platform solution compatible with different operating systems.

**Keywords:** Agriculture, traceability, blockchain, serverless, SIGPAC, Flutter, cross-platform application.

# Índice general

# Índice de figuras

# Índice de tablas

# Glosario

**Backend** Parte de una aplicación que gestiona la lógica, procesamiento de datos y acceso a la base de datos. No es visible para el usuario final.

**Bajo demanda** Modelo de provisión de recursos o servicios que se activa solo cuando el usuario lo requiere, optimizando costes y uso eficiente de la infraestructura.

**Billetera** Herramienta software o hardware que permite almacenar, enviar y recibir criptomonedas de manera segura.

**Blockchain** Tecnología de registro distribuido que permite almacenar datos de forma descentralizada, segura y resistente a manipulaciones.

**Bridges** Protocolos que permiten la transferencia de activos y datos entre diferentes blockchains.

**Caché** Memoria intermedia de alta velocidad que almacena datos temporalmente para acelerar su acceso posterior.

**Cloud Computing** Modelo de prestación de servicios informáticos (almacenamiento, servidores, bases de datos, etc.) a través de Internet bajo demanda.

**Cookies** Pequeños archivos que un sitio web almacena en el dispositivo del usuario para guardar información sobre su actividad o preferencias.

**Crash** Fallo repentino de un programa o sistema que provoca su cierre inesperado.

**Embedder** Componente que actúa como puente entre el sistema operativo y el motor de Flutter, permitiendo que las aplicaciones se integren con plataformas específicas.

**Endpoint** Punto final de comunicación en una API o servicio web, al que se pueden enviar peticiones para obtener o enviar datos.

**Faucet** Plataforma que distribuye pequeñas cantidades de criptomonedas gratuitas para fines de prueba o promoción.

**Framework** Conjunto de herramientas y bibliotecas que proporcionan una estructura común para el desarrollo de software.

**Hash** Valor numérico generado a partir de un conjunto de datos mediante una función, que sirve para verificar integridad o como identificador único.

**Hot Reload** Funcionalidad de Flutter que permite aplicar cambios en el código fuente de una aplicación de forma instantánea sin perder su estado actual.

**Hotkey** Tecla o combinación de teclas que ejecuta una acción específica en un programa de forma rápida.

**Inmutable** Característica de los datos que no pueden ser modificados una vez registrados, especialmente relevante en blockchain.

**Ledger** Registro o libro mayor digital donde se anotan transacciones de forma ordenada y segura, fundamental en blockchain.

**Multiplataforma** Software o sistema que puede ejecutarse en diferentes sistemas operativos o dispositivos sin modificaciones importantes.

**Ortofoto** Imagen aérea corregida geométricamente para que tenga la misma escala en toda su extensión, permitiendo mediciones precisas de distancias y superficies.

**Pooling** Técnica que gestiona un conjunto de recursos reutilizables (como conexiones a bases de datos) para optimizar el rendimiento.

**Relayer** Billetera que realiza transacciones en nombre de muchos usuarios.

**Rollup** Solución de escalado en blockchain que agrupa múltiples transacciones en una sola, reduciendo costes y aumentando capacidad.

**Serverless** Modelo de computación en la nube donde el proveedor gestiona automáticamente la infraestructura, permitiendo al desarrollador centrarse en el código.

**Smart Contract** Programa autoejecutable que se almacena en una blockchain y se activa automáticamente cuando se cumplen ciertas condiciones.

**Smart Farming** Aplicación de tecnologías digitales para optimizar la producción agrícola, mejorar la eficiencia y reducir el impacto ambiental.

**Trazabilidad** Capacidad de rastrear el historial, ubicación y uso de un producto o dato a lo largo de su ciclo de vida.

**Troubleshooting** Proceso de identificación y resolución de problemas o fallos técnicos en un sistema.

**Widget** Unidad básica de construcción en Flutter que representa un componente visual o funcional de la interfaz de usuario.

**Zona de Disponibilidad** Ubicación geográfica dentro de una región de un proveedor cloud, compuesta por uno o más centros de datos independientes en términos de energía, red y conectividad.

# Acrónimos y Siglas

**ABI** Application Binary Interface

**AoT** Ahead of Time

**API** Application Programming Interface

**ASP** Active Server Pages

**AWS** Amazon Web Services

**CCP** Certified Cloud Practitioner

**CLI** Command Line Interface

**CORS** Cross-Origin Resource Sharing

**ECDSA** Elliptic Curve Digital Signature Algorithm

**EC2** Elastic Compute Cloud

**EVM** Ethereum Virtual Machine

**FEGA** Fondo Español de Garantía Agraria

**GPL** General Public License

**GSI** Global Secondary Index

**HTTP** HyperText Transfer Protocol

**IaaS** Infrastructure as a Service

**IAM** Identity and Access Management

**IDE** Integrated Development Environment

**IDC** International Data Corporation

**IGN/CNIG** Instituto Geográfico Nacional / Centro Nacional de Información Geográfica

**INE** Instituto Nacional de Estadística

**IoT** Internet of Things

**JiT** Just in Time

**JVM** Java Virtual Machine

**JSON** JavaScript Object Notation

**LOPD** Ley Orgánica de Protección de Datos

**MAPA** Ministerio de Agricultura, Pesca y Alimentación

**MICT** Ministerio de Industria, Comercio y Turismo

**NIST** National Institute of Standards and Technology

**ODS** Objetivos de Desarrollo Sostenible

**OGC** Open Geospatial Consortium

**PaaS** Platform as a Service

**PIB** Producto Interior Bruto

**PoS** Proof of Stake

**PoW** Proof of Work

**RCU** Read Capacity Unit

**REST** Representational State Transfer

**RGPD** Reglamento General de Protección de Datos

**S3** Simple Storage Service

**SaaS** Software as a Service

**SDK** Software Development Kit

**SQL** Structured Query Language

**SSM** Systems Manager

**SVG** Scalable Vector Graphics

**TFG** Trabajo de Fin de Grado

**UML** Unified Modeling Language

**UPV/EHU** Universidad del País Vasco / Euskal Herriko Unibertsitatea

**VM** Virtual Machine

**WCU** Write Capacity Unit

**WKT** Well-Known Text

**WMS** Web Map Service

# 1. Introducción

La agricultura es uno de los pilares del sector primario, y pese a tantos avances que se están realizando en sistemas informáticos complejos, el campo necesita soluciones reales para sus problemas. Se dan pasos hacia la digitalización, pero este proceso es lento y todavía no tiene una adopción masiva. Más concretamente, cómo explica el artículo de Wolfert S. [1], la normalización del *smart farming* puede ayudar a separar la brecha entre grandes corporaciones y agricultores pequeños. De la misma manera, el Ministerio de Agricultura, Pesca y Alimentación (MAPA) [2] en su artículo *La transformación digital en la agricultura Española* habla sobre ello:

> El sector en su conjunto adolece de una visión coordinada que facilite la adopción masiva. Esto se agrava por la fragmentación del sector, con una **alta proporción de explotaciones pequeñas y medianas** que carecen de los recursos necesarios para implementar una estrategia digital integral.

De este punto surge **AgroFind** (agricultura + encontrar en inglés), que se centra en la creación de una aplicación multiplataforma destinada a agricultores, cuyo propósito es facilitar el registro digital de los eventos agrícolas. Está pensada para que los usuarios puedan documentar fácilmente el ciclo completo de sus cultivos, desde las fases iniciales hasta la recolección final, permitiendo así un seguimiento detallado.

La aplicación ha sido diseñada teniendo siempre en cuenta las necesidades del entorno rural, prestando especial atención a la simplicidad de uso para que los usuarios tengan más incentivo en utilizarla. Para ello, se ha priorizado una **interfaz intuitiva** y un funcionamiento que minimiza la dependencia de infraestructuras externas en lo máximo posible. Por ello, se ha optado por una arquitectura ***serverless*** o sin servidor, que evita el uso de servidores propios, reduce la complejidad de mantenimiento y delega los posibles fallos a proveedores muy establecidos en el mercado.

Además, el sistema emplea tecnología *blockchain* (sección 3.3) para registrar de forma inmutable y transparente los datos asociados a cada cultivo, reforzando así la confianza en la trazabilidad. También se ha incorporado la integración con el Sistema de Información Geográfica de Parcelas Agrícolas para la Política Agraria Comunitaria (SIGPAC, sección 2.1), lo que permite a los agricultores seleccionar y asociar sus parcelas directamente a través de referencias geográficas oficiales. Finalmente, el uso del framework Flutter posibilita que la aplicación funcione de manera multiplataforma, permitiendo su uso en dispositivos Android, iOS, Linux, macOS y Windows, pero al tener tiempo limitado, se ha centrado el desarrollo en Android (capítulo 6) pese a haber hecho pruebas en otras plataformas (capítulo 7).

En resumen, este trabajo plantea una solución realista y tecnológicamente avanzada, orientada a mejorar la digitalización del sector agrario, para que los consumidores tengan información más fiable sobre sus alimentos.

**AgroFind: Agricultura y tecnología unidas de la mano.**

## 1.1. Objetivos del proyecto

El objetivo principal de este proyecto es desarrollar una aplicación que permita a los agricultores llevar un **control digital** tanto de sus parcelas como de su actividad, registrando **desde la siembra hasta la cosecha**. Para ello, se pretende implementar un **sistema de trazabilidad basado en tecnología blockchain**, con el fin de generar un registro fiable, verificable y no modificable sobre cuándo, dónde y qué se recolecta.

Además de este objetivo central, el trabajo tiene otros fines a cumplir:

- **Hacer más accesible la digitalización del campo**. La aplicación está diseñada para personas con poca experiencia tecnológica, por lo que se ha priorizado una interfaz sencilla, clara y funcional. El objetivo es que cualquier agricultor pueda utilizarla sin necesidad de formación previa.
- **Ofrecer una trazabilidad completa de la cosecha**. A través de blockchain, se pretende garantizar que cada vez que un agricultor registra una cosecha, quede constancia de los datos clave (fecha, ubicación, cultivo, etc.) de forma permanente y segura. Esto puede ser útil tanto para el propio agricultor como para el consumidor, puesto que sabe a ciencia cierta la procedencia de lo que consume.
- **Reducir la necesidad de infraestructura compleja**. Se quiere evitar la dependencia de servidores propios, desarrollando una solución que funcione principalmente de forma local y que utilice solo servicios externos cuando sea necesario (por ejemplo, para registrar datos en la blockchain o reclamar nuevas parcelas), lo que contribuye a simplificar el mantenimiento y a mejorar la privacidad.
- **Fomentar el uso de tecnologías digitales en el sector agrario**. El proyecto tiene como meta ofrecer una solución realista, adaptada a las condiciones del entorno rural, y que sirva como puerta de entrada para que más agricultores tanto pequeños cómo medianos adopten herramientas digitales que les hagan más productivos en su día a día.
- **Desarrollar una prueba de concepto funcional**. Este proyecto se plantea como una demostración de viabilidad técnica y de concepto, por tanto, no se trata de un producto final listo para su despliegue masivo, sino de un prototipo completamente funcional. Para el transcurso del proyecto *se asume que los agricultores que utilizan la aplicación reclaman solo sus parcelas*, ya que controlar esto conllevaría implementar medidas que están fuera del alcance del trabajo (véase sección 8.3).

## 1.2. Influencia de los ODS

Los Objetivos de Desarrollo Sostenible (ODS) son una iniciativa impulsada por las Naciones Unidas [1] que busca abordar los principales retos sociales, económicos y ambientales del planeta. Estos fueron adoptados en el año 2015, y marcan una hoja de ruta para conseguir un desarrollo de la sociedad más justo, equilibrado y sostenible.

AgroFind se alinea con varios de estos objetivo s gracias a su enfoque en la digitalización del trabajo agrícola, la trazabilidad de las cosechas y la sostenibilidad del entorno rural. A continuación, se explican los ODS más relevantes para esta aplicación:

- **ODS 8: Trabajo decente y crecimiento económico.**
  AgroFind promueve el uso de una nueva herramienta digital entre agricultores, ayudando a modernizar su actividad y facilitar la gestión de sus explotaciones. Esto puede llevar a un trabajo más eficiente, organizado y con mayor fiabilidad, contribuyendo así al crecimiento económico del sector agrícola.
- **ODS 9: Industria, innovación e infraestructura.**
  La aplicación introduce una innovación tecnológica al utilizar *blockchain* para registrar de forma segura y verificable la información de las cosechas. Además, al funcionar sin necesidad de servidores propios, se simplifica la infraestructura requerida.
- **ODS 11: Ciudades y comunidades sostenibles.**
  AgroFind refuerza el tejido rural al ofrecer a los agricultores una herramienta adaptada a sus necesidades, que les permite digitalizar su actividad de forma sencilla. Al facilitar la trazabilidad y la visibilidad de la producción local, también contribuye al desarrollo sostenible del medio rural.
- **ODS 12: Producción y consumo responsables.**
  Gracias a la posibilidad de registrar parcelas, cultivos y eventos asociados a cada actividad agrícola, la aplicación permite un mejor seguimiento de la cosecha. La trazabilidad registrada utilizando *blockchain* hace que el proceso sea más transparente y confiable para los consumidores.
- **ODS 15: Vida de ecosistemas terrestres.**
  AgroFind contribuye a una gestión más sostenible del uso del suelo agrícola, permitiendo a los agricultores planificar y documentar su actividad con mayor precisión. Esto puede ayudar a evitar la sobreexplotación de ciertas áreas y mejorar el control de las rotaciones de cultivo [2].

[1] Página oficial de los ODS, https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/

[2] La rotación de cultivos es la práctica de plantar diferentes cultivos secuencialmente en la misma parcela para mejorar salud del suelo, optimizar los nutrientes en el suelo y combatir las plagas y la aparición malezas. Fuente: https://rodaleinstitute.org/es/why-organic/organic-farming-practices/crop-rotations/

### 1.2.1. Principios y valores democráticos

Al ser un proyecto hecho en la Universidad del País Vasco / Euskal Herriko Unibertsitatea (UPV/EHU), tiene que seguir el reglamento de convivencia hecho por la misma [3] y la *EHUagenda 2030* [4]. AgroFind está alineado con los mismos, entre los que destacan dos puntos del artículo 1.2:

- *Promover el respeto a la diversidad, la tolerancia, la igualdad y la inclusión.*
  La aplicación está diseñada para ser accesible a agricultores de diversas regiones y contextos, fomentando la inclusión digital y reduciendo la brecha tecnológica en el ámbito rural.
- *Fomentar la transparencia y la honestidad en el desarrollo de la actividad académica.*
  La implementación de la blockchain en AgroFind garantiza la trazabilidad y transparencia de los datos agrícolas, promoviendo prácticas honestas y verificables.

## 1.3. Encuesta de motivación

Con el objetivo de comprender mejor la percepción de los consumidores en relación con la trazabilidad de los productos agrícolas, el valor atribuido a la producción local y conocimientos generales sobre blockchain y criptomonedas, se ha realizado una encuesta anónima (véase anexo 8.3) utilizando **Google Forms**, en la que han contestado **119 personas** entre los días *2-5 de junio de 2025*. En ella, han participado consumidores de un rango de edades muy variado (figura 1.1), entre los que hay encuestados que viven en ciudades grandes y otros que residen en municipios más agrícolas, en el que todos los gráficos que se presentan en esta sección son generados por el formulario.

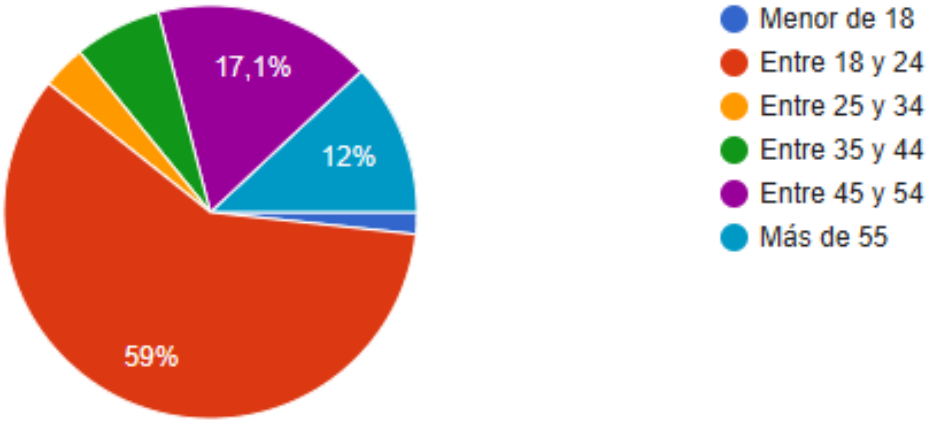



**Figura 1.1:** *Gráfico de sectores sobre la edad de los encuestados. ¿Cuál es tu edad?*

Se les ha cuestionado sobre si conocen el término de *blockchain*, y ante ello, el 41 % ha dicho que no lo ha escuchado nunca, pero la totalidad de los encuestados si han escuchado hablar de las criptomonedas (figura 1.2). Precisamente esto es lo que busca este proyecto, dar visibilidad a la tecnología que hay detrás, y que se puede emplear para muchas otras tareas, no solo para especulación o intercambio de divisas.

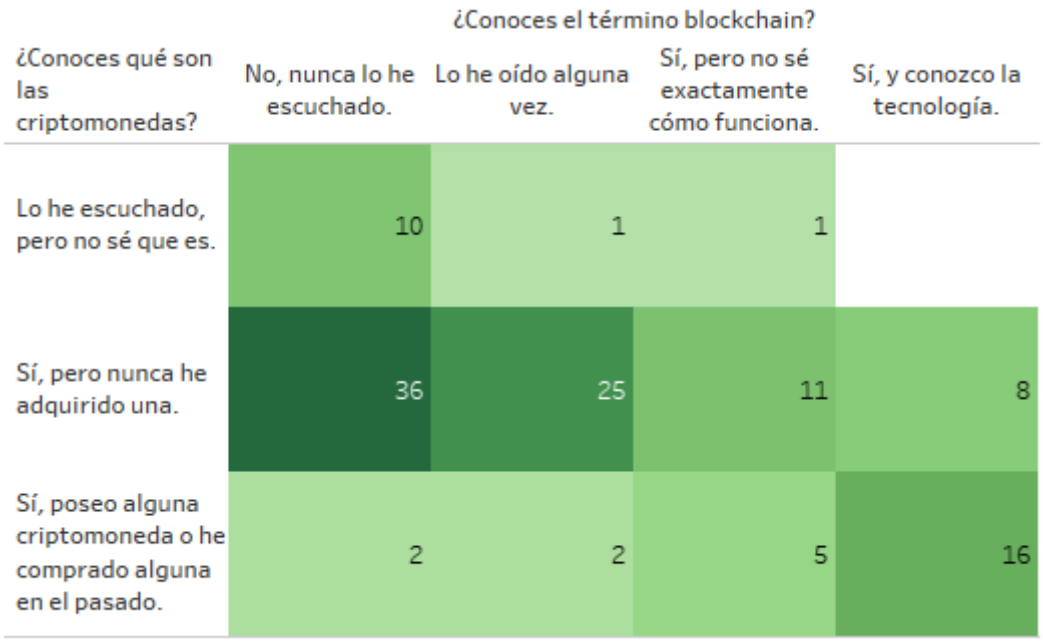



**Figura 1.2:** *Mapa de calor que cruza el conocimiento sobre criptomonedas y blockchain. ¿Conoces el término blockchain? y ¿Conoces qué son las criptomonedas?*

Respecto a la parte más monetaria, los consumidores están dispuestos a pagar más por un producto si es local, donde en la figura 1.3 destaca que el **56,4 % desembolsaría entre un 6 y un 20 % más por el mismo producto** siendo éste más cercano.

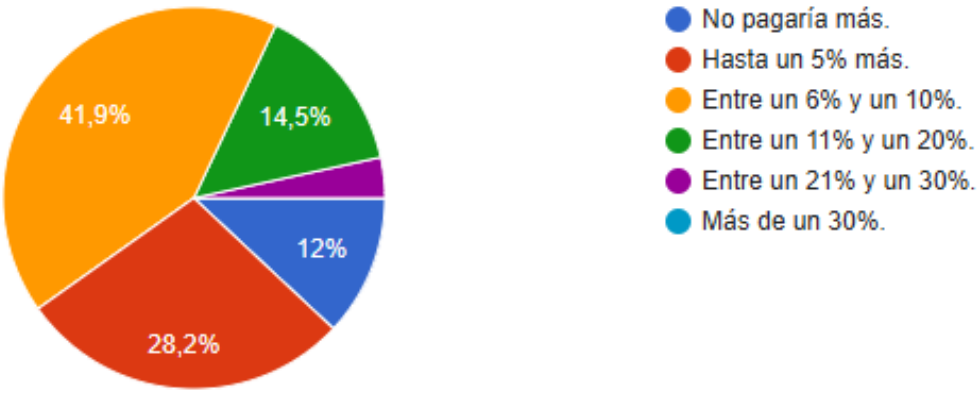



**Figura 1.3:** *Gráfico de sectores.*
*¿Cuánto estarías dispuesto/a a pagar por un producto local en comparación a uno extranjero?.*

## 1.4. Razones de elección del trabajo

El autor ha propuesto la idea de este TFG ante el tutor por varias razones:

La primera es demostrar que se pueden hacer proyectos muy útiles utilizando la blockchain pese a que se le ha dado una fama muy mala recientemente con las criptomonedas, porque estas son solo una de las muchísimas aplicaciones que tiene esta tecnología. Esto no es nuevo, la cadena de supermercados Carrefour empezó a utilizar una blockchain privada en 2022 para trazar sus productos *Bio* [5].

La segunda es la convicción de ayudar al campo. El mundo agrícola en España está pasando por momentos muy delicados, puesto que cada vez se consume más producto extranjero y esto hace que los labradores pequeños y medianos tengan más dificultades en ser rentables, según el Ministerio de Industria, Comercio y Turismo (MICT) en la balanza comercial de 2022 [6]. Cada año que pasa, menos personas se quieren dedicar al campo (véase sección 2.1), y si no se intenta renovar utilizando las herramientas tecnológicas ya existentes, no va a ser atractivo para futuros agricultores.

La tercera es querer poner en práctica todo lo aprendido en las prácticas de empresa. En estas se ha aprendido sobre el uso de Amazon Web Services (AWS) para un proyecto real, y de cómo se puede tener funcionando una aplicación a gran escala sin tener servidores directamente, utilizando distintas herramientas. Hay que tener en cuenta que el autor posee el título AWS Certified Cloud Practitioner [3], y este conocimiento ha sido indispensable para el desarrollo de AgroFind.

Se usará como ejemplo a lo largo de la memoria un municipio de la provincia de Burgos llamado Medina de Pomar, que incluye varias localidades dentro de sí. Concretamente, la ubicación se fijará en **Moneo** [4], un pueblo pequeño (figura 1.4), rodeado de parcelas, el cual es un ejemplo perfecto para el uso de AgroFind, donde varios agricultores tienen parcelas colindantes a poca distancia. Esta es otra de las razones del proyecto, evitar que pueblos así desaparezcan por la renovación generacional.


**Figura 1.4:** *Imagen aérea del pueblo de Moneo. Fuente:* https://www.google.es/intl/es/earth/

[3]Enlace al título: https://www.credly.com/badges/fbc4c49e-c38a-4aa4-bbae-12cf189831ec

[4]Moneo, localidad Burgalesa situada en Las Merindades, antiguamente conocida como Aforados de Moneo (véase [7]) es el pueblo natal de parte de la familia del autor.

## 1.5. Estructura de la memoria

La memoria está organizada en ocho capítulos, cada uno de los cuales aborda un aspecto clave del desarrollo del proyecto, además del resumen, índice de tablas, índice de figuras y la bibliografía. A continuación, se describe brevemente el contenido de cada uno:

- **Capítulo 1: Introducción**
  Se presenta el proyecto, los objetivos del mismo, la motivación personal para su desarrollo, la relación con los ODS y su alineación con los principios y valores democráticos de la UPV/EHU.
- **Capítulo 2: Contexto del Dominio**
  Se analiza el sector agrícola en España y el uso del sistema SIGPAC como herramienta clave para la geolocalización de parcelas. Este contexto sirve de base para entender la utilidad de la aplicación.
- **Capítulo 3: Contexto Tecnológico**
  Se revisan las tecnologías empleadas en el desarrollo del proyecto: desarrollo multiplataforma con Flutter, servicios en la nube, y blockchain. Se describen las herramientas utilizadas y su aplicación dentro de AgroFind.
- **Capítulo 4: Planificación**
  Se detallan los objetivos técnicos del proyecto, las herramientas utilizadas, el alcance funcional, la planificación temporal, una estimación económica y un análisis de riesgos.
- **Capítulo 5: Análisis y Diseño**
  Se definen los requisitos funcionales y no funcionales, los casos de uso, el modelo de dominio, y la arquitectura general de la aplicación, incluyendo el cliente y una visión del backend.
- **Capítulo 6: Implementación**
  Se describe el proceso de desarrollo de la aplicación en Flutter, incluyendo la estructura del código, las pantallas principales y la integración con tecnologías como Firebase, AWS, las distintas APIs y la blockchain.
- **Capítulo 7: Pruebas**
  Se incluyen tanto las pruebas técnicas (unitarias y automatizadas) como las pruebas en distintos dispositivos y sistemas operativos. Además, se detallan las pruebas de usuario realizadas, su metodología y los resultados obtenidos.
- **Capítulo 8: Conclusiones**
  Se recogen las conclusiones generales del proyecto, el grado de cumplimiento de los objetivos, posibles desviaciones temporales y económicas, valoración personal del trabajo y propuestas de mejora de cara al futuro.

# 2. Contexto del Dominio

Históricamente, la agricultura ha sido una parte importante del desarrollo económico y social, pero más concretamente, a día de hoy en el contexto del siglo XXI, su relevancia se mantiene tanto por su contribución a la economía como por el impacto que sigue teniendo. Este trabajo está relacionado fuertemente con este ámbito, por tanto, es pertinente hacer una introducción al mundo agrícola en el país y las iniciativas que surgen por aprovechar la tecnología para hacerlo más eficiente.

## 2.1. La agricultura en España

El sector agrícola en España representa uno de los pilares fundamentales del sector primario, actualmente contribuyendo un **2.34 % al Producto Interior Bruto (PIB)** en base a los datos de Statista [8]. De hecho, según los datos del Parlamento Europeo y del Banco Mundial [9], España es el cuarto país que más empleo genera en el ámbito agrícola en la Unión Europea, superando a Francia y Alemania [10].

En contraposición a la importancia el sector, según el Instituto Nacional de Estadística (INE) **el número de explotaciones agrícolas en España se redujo un 12,4 % en 2023 respecto a 2020** [11], debido a una tendencia continuada de concentración de tierras y abandono progresivo de la actividad por parte de explotaciones familiares y pequeñas fincas.

Hay múltiples causas de este fenómeno, donde destacan el envejecimiento de los agricultores, la competencia internacional, las exigencias medioambientales y el más influyente en esta década, la pandemia, que llevó a muchos pequeños productores a cesar su actividad económica. Todo ello contribuye a una creciente precariedad en el sector, haciendo más difícil la llegada de agricultores jóvenes con expectativas de innovar.

Además de ello, citando textualmente al análisis [12] del **MAPA**:

> A nivel nacional la Superficie Agraria Útil en régimen de propiedad alcanza 12,4 mill. de hectáreas (52 % del total), mientras que en régimen de arrendamiento se cultivan 8,9 mill. de hectáreas (37 % del total).[a]
>
> ---
>
> El envejecimiento de la población agraria es un hecho generalizado en todas las comunidades autónomas. Es especialmente destacable que el 41 % de los jefes de explotación tiene más de 65 años, mientras que sólo el 8 % tiene menos de 40. [b]
>
> [a] *Sección 2.2.* Distribución de la superficie agraria útil por régimen de tenencia.
> [b] *Sección 2.4.* Caracterización de los/as jefes/as de explotación.

Respecto a la primera cita, el campo español tiene un factor importante de agricultores que alquilan terrenos debido al alto coste de los mismos, y por tanto, el cambio de dueño de las parcelas tiende a pasar con menos frecuencia, mientras que el cambio del usuario que las explota sí es más común.

Sobre el punto 2.4 del análisis del MAPA [5], esta tendencia de edad en los agricultores hace que realizar un uso beneficioso de las nuevas tecnologías sea más complicado. Sin embargo, no por ello se han dejado de crear propuestas para renovar el campo español. Tanto en el sector público como en el privado han surgido proyectos nacionales con el objetivo de modernizar las explotaciones y facilitar la transición digital.

Entre ellas se pueden destacar iniciativas como *Ebro Valley Agro Tech* [13], que es un evento organizado por el Ministerio de Transformación Digital y varias startups agrícolas de toda la península para intentar consolidar relaciones entre las mismas. Del ámbito más de *hardware*, *Smart Agro* [14] es una solución desarrollada por *Odin Solutions*, que es plataforma tecnológica orientada a implementar dispositivos *Internet of Things (IoT)* en explotaciones privadas, permitiendo el control de parámetros como la humedad, la temperatura o el estado de los cultivos. Por último, cabe mencionar *SIGPAC*, que se desarrolla en profundidad en la siguiente sección y es de vital importancia para este trabajo.

Finalmente, un dato a tener en cuenta es la confianza de los consumidores sobre los productos que salen del sector, ya que el tener un sistema de trazabilidad puede ayudar a mejorar este dato. Según el *barómetro del clima de confianza del sector agroalimentario* [15] que realiza el MAPA, el origen de los productos que consume le importa mucho o bastante al 87,6 % de los encuestados y a su vez, según el EIT Food Trust Report de 2023 [16], España es el país europeo en el que sus habitantes se fían más de los agricultores, pero los consumidores confían entre un 15 y un 20 % menos en las grandes superficies (supermercados e hipermercados principalmente) a la hora de comprar productos de alimentación frescos.

[5] Ministerio de Agricultura, Pesca y Alimentación, enlace: https://www.mapa.gob.es/es/default.aspx

## 2.2. SIGPAC

Con el fin de utilizar la tecnología disponible para el beneficio de los agricultores, en 2005 el MAPA puso en vigor el *Sistema de Información Geográfica de Parcelas Agrícolas para la Política Agraria Comunitaria*, mejor conocido como SIGPAC, respondiendo así a la exigencia de la Unión Europea de que cada Estado miembro disponga de un Land Parcel Identification System (LPIS) [17], siendo el SIGPAC la implementación nacional de dicho sistema [18].

Gracias a este sistema, los agricultores pueden localizar sus terrenos, ya sean parcelas o de otro tipo, mediante el visor online que ofrece el Fondo Español de Garantía Agraria (FEGA) junto con el MAPA [19], en el que se incorpora información catastral, límites administrativos, superficies cultivables, usos del suelo y otras capas de interés general para el labrador. Como se puede ver en la figura 2.1, el usuario del visor puede seleccionar que capas (mapa forestal, recintos, pastos permanentes...) ver en el mapa, consultar información sobre la parcela pinchando en esta y además puede elegir si desea ver el mapa de la pasada campaña de cosechas o el de este año.

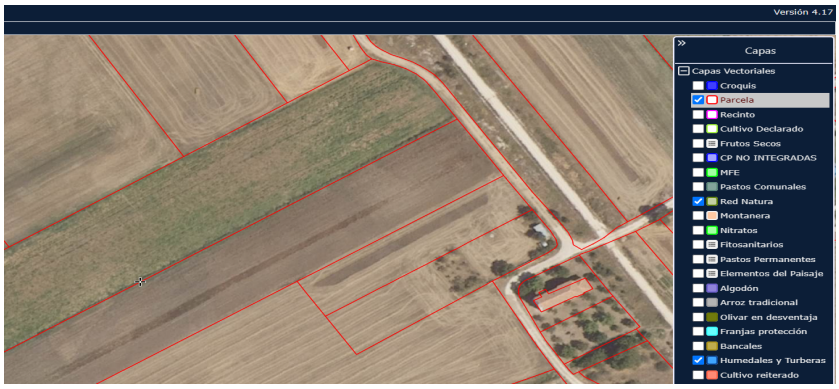



**Figura 2.1:** *Captura de pantalla del visor que ofrecen el FEGA y el MAPA.*

En términos más técnicos, SIGPAC está compuesto por una base de datos construida a partir de ortofotos, que son **imágenes aéreas corregidas geométricamente** y tomadas normalmente mediante satélites y avionetas. Estas imágenes permiten a los agricultores ubicar las parcelas mediante la posición que ocupan en el paisaje y además, se **actualizan anualmente** para reflejar los cambios en el uso del suelo que hayan podido suceder.

Otro de los aspectos de SIGPAC es que ofrece varias herramientas desde las que se puede obtener esa misma información de manera más precisa mediante APIs (Application Programming Interfaces), servicios WMS (Web Map Service) y similares, siguiendo la directiva INSPIRE de 2007 [20]. Los manuales de uso de todos ellos se encuentran en SIGPAC Hubcloud [6], de los que para este trabajo se han empleado algunos de ellos, entre los que destacan los siguientes:

- **Servicio WMS**. Para que los agricultores puedan elegir cuáles son las parcelas en las que desempeñan su labor sin tener que dibujarlas se emplea esta herramienta [21] que permite obtener imágenes de los límites (lindes) de las parcelas

[6]Enlace a los manuales de uso: https://sigpac-hubcloud.es/index.html

e información sobre ellos mediante el protocolo definido por el Open Geospatial Consortium [7], a través de peticiones HyperText Transfer Protocol (HTTP) con parámetros específicos como `GetFeatureInfo` y `GetMap`.
Del primero se puede sacar información concreta sobre uno o varios recintos, mientras que del último se obtiene la capa *AU.Sigpac:recinto* que permite visualizar las imágenes y cargarlas en un mapa [22].

- **Propiedades de recinto por coordenadas**. Dadas unas coordenadas, el servicio [23] devuelve información sobre las propiedades del recinto, que son de gran utilidad para el agricultor, entre las que se encuentran por ejemplo, el identificador único para cada parcela que se comentará su uso posteriormente, la utilidad SIGPAC del terreno, el coeficiente de regadío, las hectáreas del terreno, y las coordenadas en formato Well Known Text (WKT)[8]. Estas últimas permiten dibujar en un mapa el recinto a escala de manera directa.

La implementación técnica de estos dos servicios se integra en el capítulo 6. Además, es importante mencionar que todos los servicios utilizados de SIGPAC [9] tienen licencia *Creative Commons* 4.0 y por tanto, son libres de su uso tanto directo como adaptado, que es el objeto de este trabajo.

### 2.2.1. Antecedentes

Finalmente, hay antecedentes a destacar que utilizan el servicio del MAPA con distintos fines, desde ganadería hasta cartografía pasando por protocolos de inundaciones:

- **Cartografía de cultivos mediterráneos**. Realizado por la UNED [24] en 2008, este artículo usa la base de datos SIGPAC para cartografiar los cultivos y hacer un enriquecimiento vectorial de los datos para hacer que los polígonos de las parcelas sean lo más fieles posibles a la realidad.
- **Cumplimiento de legislación ganadera**. La iniciativa [25] explicada en la revista de ganadería de la Editorial Agrícola Española (2008) comenta como el uso de este sistema ayuda a los ganaderos de cerdos ibéricos a cumplir con las nuevas regulaciones y a gestionar más eficientemente su ganado.
- **Clasificación de usos de suelo para aplicaciones hidrológicas**. Este artículo [26] hecho en la Universidad de Sevilla durante 2010 usa una estrategia híbrida entre SIGPAC y LANSAT, permite clasificar de forma más precisa los usos del suelo, que orienta a mejorar el análisis y modelización de procesos hidrológicos.
- **Mejora de protocolo sobre inundaciones**. Un trabajo de fin de máster más reciente (2021) [27] propone el uso de SIGPAC junto con tecnologías LiDAR para mejorar el protocolo de los análisis que se realizan sobre los riesgos de inundación en los campos.

[7] Open Geospatial Consortium (OGC), enlace: https://www.ogc.org/
[8] Visor de archivos WKT, enlace: https://wktmap.com/
[9] SIGPAC - MAPA. Licencia: https://creativecommons.org/licenses/by/4.0/

# 3. Contexto Tecnológico

Respecto al ámbito directamente relacionado con la informática, a continuación se ponen en contexto las diferentes tecnologías, conceptos y herramientas que se han utilizado para el trabajo, detallando el estado del arte y antecedentes de las mismas.

## 3.1. Aplicaciones multiplataforma

Al ser una aplicación principalmente móvil, debido a que es más sencillo para un agricultor usarla desde el teléfono y no desde un ordenador, se podría desarrollar **nativamente**, pero esto causa un problema y es que según StatCounter [28] el 74.52 % de los usuarios en España utiliza **Android**, mientras que el 25.13 % se decanta por **iOS**. Por tanto, si se desarrollase una app nativa de Android, un cuarto de los ganaderos no podría usar la app. Para evitar esto hay dos opciones:

Construir una aplicación híbrida consiste en crear una solución web y adaptar la pantalla al el dispositivo, pero es poco eficiente y no tiene acceso a muchas las funciones específicas de cada Sistema Operativo, mientras que una aplicación multiplataforma es más cercana a una nativa, sin tener que programar dos códigos diferenciados [29].

Esta se desarrolla para poder ejecutarse en diferentes sistemas operativos a la vez que en distintos tipos de dispositivos, partiendo de una única base de código. Esto permite escribir la app una sola vez y desplegarla en varias plataformas sin tener que volver a programar por separado. El precursor de este enfoque fue Oracle con la Java Virtual Machine (JVM) [30] que transforma el código Java a lenguaje máquina, lo compila y lo ejecuta, donde cada aplicación o módulo tiene su propio JVM [31]. Se verá en la sección 3.3 que el funcionamiento de la Ethereum Virtual Machine (EVM) está inspirado en la JVM en muchos aspectos.

Gracias a este enfoque se reducen notablemente los costes tanto temporales como de desarrollo al mantener solo una versión del programa, y también facilita el mantenimiento de la aplicación, porque las actualizaciones o mejoras se aplican sobre una misma base de código. Entre las tecnologías más comunes para desarrollo multiplataforma destacan **Flutter, React Native, .NET MAUI y Kotlin**. A continuación (figura 3.1) se presenta una tabla comparativa entre las opciones que se han planteado elegir para realizar *AgroFind* con los datos a mayo de 2025:

| | Flutter | React Native | .NET MAUI | Kotlin |
|---|---|---|---|---|
| **Estrellas** ∗ | 170k | 122K | 22.7k | 50.5k |
| **Forks** ∗ | 28.5k | 24.6k | 1.8k | 5.9k |
| **Issues** ∗ | 105k | 26.5k | 14.2k | – |
| **Pull Requests** ∗ | 61k | 23.3k | 10.4k | 5.3k |
| **Tendencia** ◇ | 15 % ↑ | 10 % ↑ | 5 % ↓ | 0 % = |
| **Porcentaje de uso** ◇ | 9.4 % | 8.4 % | 3.1 % | 6.2 % |
| **Google Trends** ○ | Dominante | Dominante | Minoría | Estable |
| **Lanzado por** | Google | Meta | Microsoft | JetBrains |
| **Lenguaje utilizado** | Dart | JS, JSX y CSS | C# y XAML | Kotlin |

**Tabla 3.1:** *Comparativa de frameworks multiplataforma (a mayo de 2025).*

∗ Datos de GitHub actualizados a 12/05/2025. Fuentes: repositorios oficiales ([32], [33], [34], [35]).
◇ Encuesta *StackOverflow Technology* de 2024, fuente: [36]. ↑ indica crecimiento anual; ↓ decrecimiento; = indica sin cambios. El porcentaje de uso es el número de desarrolladores que emplean el framework correspondiente como parte crucial de su trabajo.
○ Google Trends de 2025 sobre los 4 frameworks de la tabla. Fuente: [37].

A raíz de la información condensada en la tabla se ha hecho otra figura dónde se elige qué tecnología es la más adecuada para el trabajo. La decisión se basa en ciertos requisitos que el autor considera oportunos para el desarrollo del proyecto, incluyendo además una preferencia personal de trabajar con cada tecnología.

Los criterios considerados son los siguientes:

- **Antigüedad**: En base al tiempo que el toolkit lleve en funcionamiento.
- **Futuro y tendencias**: Basado en las tendencias de Stack Overflow.
- **Comunidad**: Teniendo en cuenta las métricas de GitHub.
- **Integración**: Las herramientas que cada framework posee para gestionar mapas, solicitudes API, integración con Firebase [10] o AWS Cognito [11]...
- **Motivación personal**. Ganas de aprender el lenguaje correspondiente y gustos visuales sobre aplicaciones realizadas en dicho framework.

A cada opción se le asigna una puntuación de entre 1 y 5 para cada criterio. Estos valores han sido estimados manualmente a partir de la información presentada en la tabla 3.2. Cada criterio tiene asociado un factor de multiplicación para ponderar su importancia relativa.

[10] Página web de Firebase: https://firebase.google.com/
[11] Página web de AWS Cognito https://aws.amazon.com/es/cognito/

| Criterio | Flutter | React Native | .NET MAUI | Kotlin | Factor |
|---|---|---|---|---|---|
| Antigüedad | 4 | 4 | 2 | 4 | 0.1 |
| Futuro y tendencias | 5 | 5 | 2 | 4 | 0.2 |
| Comunidad | 4 | 5 | 2 | 3 | 0.2 |
| Integración | 4 | 5 | 2 | 4 | 0.3 |
| Motivación personal | 5 | 1 | 3 | 5 | 0.2 |
| **Resultado** | **4.4** | **4.1** | **2.2** | **4** | |

**Tabla 3.2:** *Evaluación ponderada de frameworks a seleccionar.*

De esta manera, se ha decidido optar por **Flutter**, aún habiendo explorado Kotlin como alternativa. Pese a la cercanía en los puntos entre el elegido y React Native para el desarrollo de la app, la experiencia del autor con JavaScript es muy pequeña, y cree que visualmente Flutter resulta más atractivo y moderno.

### 3.1.1. Flutter

Técnicamente hablando, Flutter es un kit de herramientas para interfaces de usuario que en la práctica actúa como un framework completo. Es de código abierto, diseñado por **Google** [12] y fue lanzado en 2017 con el fin de desarrollar aplicaciones multiplataforma para dispositivos móviles, web y de escritorio, con una misma base de código. Como explica AWS en su blog [38], destaca por varias razones:

- **Rendimiento casi nativo**. Flutter utiliza el lenguaje **Dart** (véase 3.1.1), que se compila a código máquina y lleva a mucha eficiencia en tiempo de ejecución.
- **Motor gráfico personalizado**. **Skia** es una biblioteca de software de gráficos diseñada específicamente, que hace que la experiencia de usuario sea más fluida. Además, recientemente se ha incluido **Impeller** como otra opción más eficiente pero con más restricciones.
- **Herramientas para desarrolladores**. Al ser un framework diseñado por Google, este posee implementaciones útiles cómo el *inspector de widgets* y la recarga en caliente o *hot-reload*.

#### Arquitectura interna

Para entender mejor como funciona Flutter, hay que explicar un poco de su arquitectura [39], que se divide en tres partes principales (figura 3.1):

Empezando por el ***embedder*** (que es un puente entre el sistema operativo y el motor de Flutter), es el encargado de proporcionar el punto de entrada a la aplicación, gestionar el ciclo de vida y da acceso a servicios como el renderizado de superficies, accesibilidad y otros concretos del SO. Está programado principalmente en *C++* y da soporte a Android, Windows, iOS, macOS y Linux, adaptándose a cada uno.

[12] Página web oficial de Flutter: https://flutter.dev/

El componente central es el ***engine*** o **motor**, que es el núcleo de todo el sistema y está también escrito en *C++*. Se encarga de interpretar, compilar y ejecutar el código Dart, de renderizar tanto los gráficos por pantalla usando Skia o Impeller como dibujar correctamente el layout, los textos, botones... También gestiona la lectura y escritura de archivos, los eventos y notificaciones del sistema y los plugins de bajo nivel, donde se encuentran tanto las notificaciones por un lado, como la cámara, geolocalización [40] y web views. Para unir el motor con Dart, se hace a través de la biblioteca `dart:ui`, que se encarga de la comunicación entre el framework y el *engine*.

Finalmente, la capa más alta en esta arquitectura es el ***framework***, que es la interfaz con la que los desarrolladores interactúan con Flutter. Si se sigue el orden de la figura 3.1, en este segmento encontramos funciones básicas y servicios visuales, además de una capa de renderizado para permitir desarrollar los árboles de objetos. Un escalón por encima, se encuentran los ***widgets***, que son la unidad básica de la construcción de la interfaz en Flutter (véase 3.1.1) y por último, librerías de alto nivel, las indispensables (Cupertino para iOS y Material para Android) y a las que añadiremos varias para este proyecto. Por mencionar algunas de las más importantes que se pueden ver en el archivo `pubspec.yaml`, `firebase_core` [13], que se encarga de integrar Firebase con la aplicación, `flutter_map` [14], cuyo propósito es gestionar el mapa y todo lo relacionado con manejarlo, dibujar polígonos e interactuar con él.

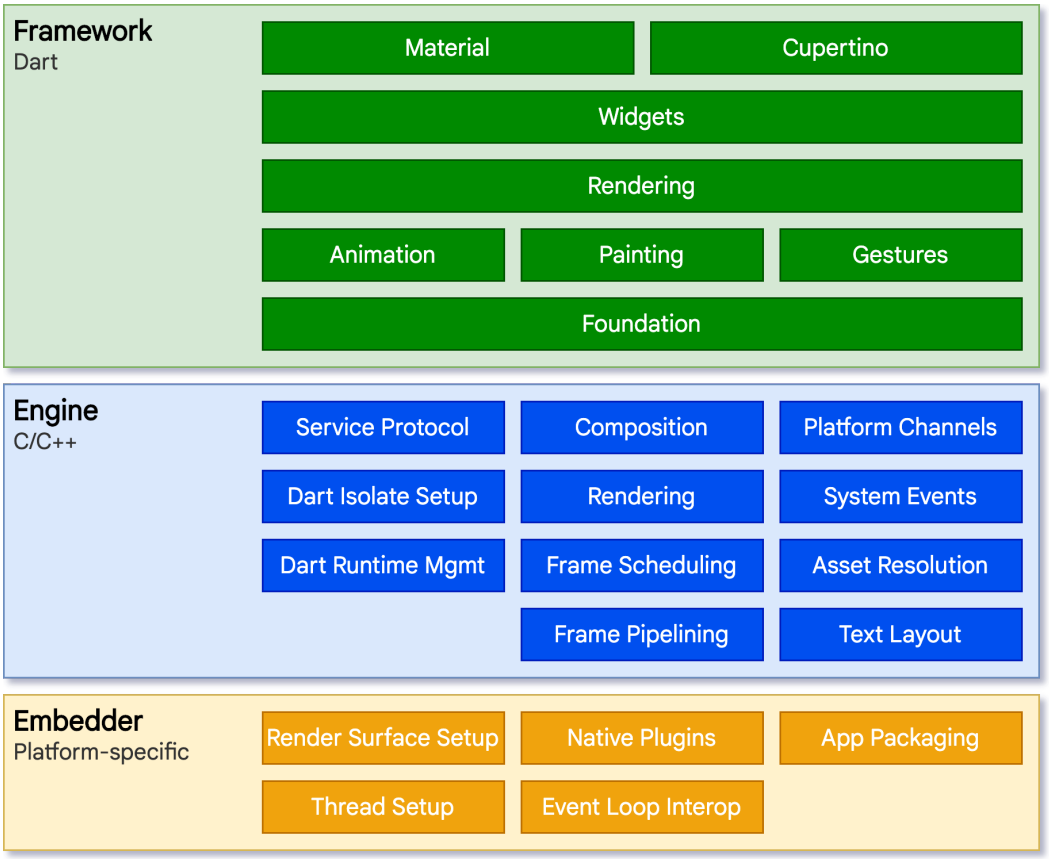



**Figura 3.1:** *Capas de la arquitectura de Flutter. Fuente: [39]*

[13] Biblioteca firebase_core, enlace: https://pub.dev/packages/firebase_core
[14] Biblioteca firebase_map, enlace: https://pub.dev/packages/flutter_map

### Widgets

Estos son una clase inmutable de Dart, que describen cómo debería de ser una parte de la interfaz de usuario en un momento en el tiempo [41]. Puede haberlos más o menos complejos, desde un `Text` que se encarga de visualizar textos, hasta `GestureDetector` que como su nombre indica detecta los gestos del usuario (deslizar, tocar, mantener pulsado...). Por defecto, hay dos tipos principales:

- **Sin estado** o `StatelessWidget`. Es *inmutable*, esto quiere decir que una vez creado, tanto su contenido, como sus estilos, como su aspecto son fijos mientras esté en pantalla. Para poder modificar la información se tiene que recrear la pantalla al completo, al igual que en la clase `LoginTextBox`, que es una caja de texto personalizada para la pestaña de iniciar sesión, y para hacer el código más eficiente, se define como un widget sin estado (extracto de código 3.1):

```dart
// librería material que contiene iconos y herramientas de diseño
import 'package:flutter/material.dart';

class LoginTextBox extends StatelessWidget {
  // atributos de entrada
  final String textoPista;
  final bool oscurecerTexto;
  // controlador de texto
  final TextEditingController controlador;
  // inicialización de los parámetros
  const LoginTextBox({
    super.key, required this.textoPista,
    required this.oscurecerTexto, required this.controlador,
  });

  @override
  Widget build(BuildContext context) {
    // devuelve el campo de texto personalizado
    return TextField(
      controller: controlador,
      decoration: InputDecoration(
        hintText: textoPista,
        border: OutlineInputBorder(
          borderRadius: BorderRadius.circular(10),
        ),
      ),
      obscureText: oscurecerTexto,
    );
  }
}
```

**Extracto de código 3.1:** *Clase LoginTextBox, ejemplo de StatelessWidget.*

- **Con estado** o `StatefulWidget`. Puede cambiar durante tiempo de ejecución sin necesidad de recrear la actividad al completo. Tiene un estado interno *mutable*, y cuando se hace algún cambio en el mismo, se llama a la función `setState()`, que actualiza la interfaz. A continuación (extracto 3.2), ejemplo de una clase que se ha usado en el proyecto, para decidir si hay que mostrar el login o el registro:

```dart
class LoginORegistro extends StatefulWidget {
  const LoginORegistro({super.key});
  @override
  State<LoginORegistro> createState() => _LoginORegistroState();
}

class _LoginORegistroState extends State<LoginORegistro> {
  bool mostrarLogin = true;
  void cambiar() {
    // función de cambio de estado
    setState(() {
        mostrarLogin = !mostrarLogin;
    });
  }
  /*  resto del código  */
}
```

**Extracto de código 3.2:** *Clase LoginORegistro, ejemplo de StatefulWidget.*

### Dart

Dart es un lenguaje de programación desarrollado por Google en 2011 [42], creado para poder hacer aplicaciones en varias plataformas con el mismo código. Tiene tipado seguro o *type safe* (extracto 3.3) por tanto, cada variable tiene que coincidir siempre con su tipo pero no es necesario escribirlo explícitamente porque se infiere. Esto no limita a que exista un tipo `dynamic` que da esa flexibilidad de cambios de tipo que ha sido muy útil durante el proyecto.

Además, incluye seguridad contra nulos (*null safety*) y esto conlleva a que las variables no sean nulas a no ser que se especifique lo contrario. Gracias a esto, se pueden evitar muchos posibles errores de compilación y hacer más sencillo el desarrollo.

```dart
class Parcela {
  int id;                              // número entero y no nulo
  String? nombre;                      // variable que puede ser nula
  Map<String, dynamic> informacion;    // tipo dinámico y no nulo
  /* resto del código */
```

**Extracto de código 3.3:** *Clase Parcela, ejemplo del tipado en Dart.*

El compilador de Dart tiene la capacidad de ejecutar el código de dos maneras distintas, en las que para ambas hay herramientas de desarrollo y de producción, entre las que destaca el **hot-reload** o recarga rápida, que permite aplicar cambios en el código sin tener que reiniciar la aplicación (siempre y cuando sean cambios que no afecten a archivos de configuración o estados de los widgets). También ofrece varias herramientas diferentes dependiendo de la plataforma para la que se quiera desplegar (figura 3.2):

- **Nativa**. Para el desarrollo de apps en Android, iOS, macOS, Windows y Linux, se compila la aplicación Just In Time (JiT) en desarrollo para habilitar el hot-reload y las características dinámicas. En cambio, para desplegarla en producción, se compila usando Ahead Of Time (AoT), que asegura tiempos de carga menores, con gestión de memoria personalizada y un recolector de basura mejorado.

- **Web**. Respecto a las páginas web, Dart permite desplegar el código tanto en *WebAssembly* para mayor rendimiento, como en *JavaScript* para mayor compatibilidad, mientras que el despliegue incremental en JavaScript es una opción adicional.

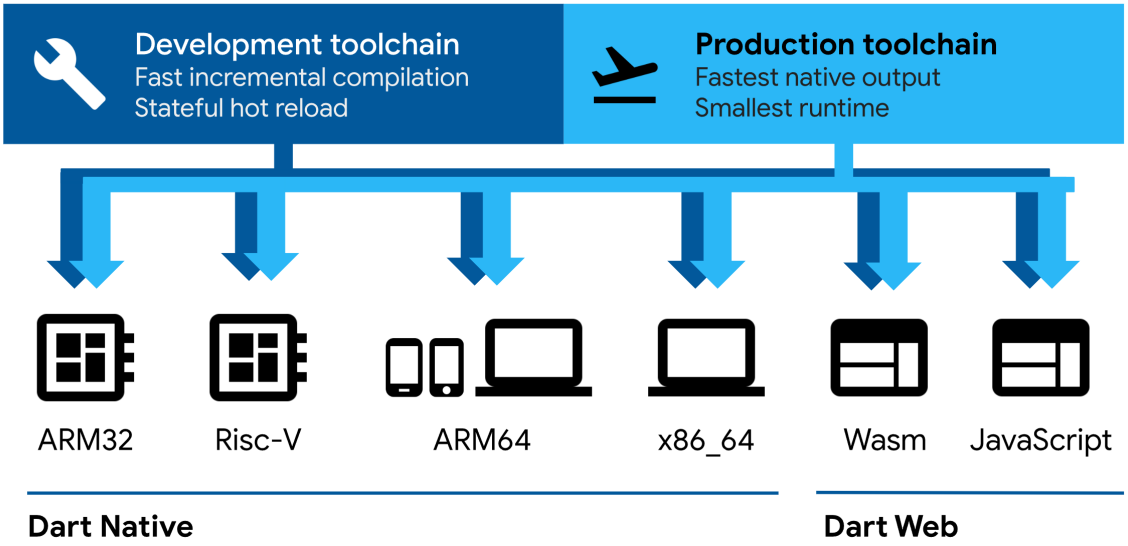



**Figura 3.2:** *Ejecución de Dart en las plataformas. Fuente:* https://dart.dev/overview

Respecto al tiempo de ejecución, Dart se ejecuta en una Máquina Virtual, a la que tiene asociada un recolector de basura y checkeo de variables, además del control de hilos y secuencias secundarias (llamados *isolates*) [43].

Según el artículo de 'Theo' Karasavvas [44], en el que explica en el blog de Stack Overflow porqué Flutter es el más popular entre los Kits de Desarrollo de Software (SDK) multiplataforma disponibles, hay que destacar que la facilidad de aprendizaje de Dart, se debe a que tiene una estructura similar a Java o C# tanto por la orientación a objetos como por la escritura del código. También da un dato, y es que Flutter hizo un sorpasso a React Native durante 2022 en su uso a nivel mundial según datos de Statista [45].

Aún así, solo con la construcción de interfaces y gestión de datos no es suficiente para hacer una aplicación completa y aquí es donde entra en juego la computación en la nube o *cloud computing*.

## 3.2. Computación en la nube

Lo que hace una década era la tecnología más disruptora hoy en día ya se considera como algo estándar para aplicaciones comerciales, herramientas internas en el mundo empresarial e incluso infraestructura pública.

### 3.2.1. Definición y características

Definir que es el *cloud computing* no es trivial, pero si hubiera que elegir una definición, la más adecuada sería la que cita Kim Won en su artículo *Cloud Computing, Today and Tomorrow* [46], que es la literatura más relevante sobre este tema:

> The simplest working definition of cloud computing is being able to access files, data, programs and third party services from a Web browser via the Internet that are hosted by a third party provider and paying only for the computing resources and services used.

Básicamente, es un modelo de procesamiento que permite a un tercero a ofrecer servicios informáticos a través de internet. De esta forma, los recursos, ya sean hardware, software o datos, se pueden ofrecer a los clientes bajo demanda (*on demand*). Esto significa que quién adquiere el servicio puede decidir qué quiere y cuándo lo quiere, además de pagar solo por lo que usa.

A efectos prácticos, la nube son **los recursos de un tercero**, normalmente (pero no necesariamente) de empresas gigantes como son Google, Amazon, Microsoft, Oracle, Alibaba... que se encargan de comprar, configurar, mantener y gestionar equipos informáticos desde servidores hasta almacenes de datos o tarjetas gráficas.

Para que un servicio se considere computación en la nube tiene que cumplir con estos requisitos según el Instituto Nacional de Estándares y Tecnología (NIST) [47]:

1. **Autoservicio bajo demanda**. El consumidor puede aprovisionar recursos sin una intervención humana directa del proveedor del servicio, y el proveedor tiene que ser capaz de entregar estos servicios a tiempo real.

2. **Acceso amplio de red**. Los servicios tienen que ser accesibles a través de la red, en plataformas estandarizadas y homogéneas.
   Nota del autor: *normalmente esto conlleva que las plataformas de cloud se gestionan a través de una interfaz web o una Command Line Interface (CLI).*

3. ***Pooling* de recursos**. Los recursos de computación del proveedor (ya sean físicos o virtuales) se comparten de manera dinámica entre múltiples dispositivos. El consumidor no tiene la ubicación exacta de los recursos, pero se recomienda dar un nivel de abstracción sobre el lugar de los mismos.

4. **Elasticidad rápida**: Tiene que haber una capacidad de escalar recursos bajo demanda, por ejemplo, si un proceso recibe mucha información, tiene que poder ampliar su poder de computación de manera automática.

Nota del autor: *el término elasticidad se suele usar con frecuencia en los servicios de los proveedores para denotar el poder de escalado bajo demanda.*

5. **Servicio medido**: Los sistemas tienen que tener un uso controlado, monitorizado y facturado automáticamente, donde se pague solo por el uso de los recursos, aunque el usuario y el proveedor puedan llegar a acuerdos de exclusividad de uso, que pueden conllevar un coste mayor al uso per se.

Sobre la historia de la computación en la nube [48] se podría hacer una división en 4 franjas de tiempo y para ello se ha hecho la figura 3.3.

Esta tecnología tuvo su concepción teórica en los años 60, cuando surgió el *time-sharing* [49], compartir de forma concurrente uno o más recursos computacionales entre muchos usuarios. En aquel momento esto se hacía mediante interrupciones o mediante procesamiento por lotes.

Hacia el final del siglo XX, aparecen los Application Service Providers o ASPs. Empresas que ofrecen un servicio o producto a través de internet. De estos nace **Salesforce** como la primera empresa de software como servicio (SaaS) y **VMWare** con la virtualización que es un aspecto clave de la nube.

Finalmente en 2006, **Amazon Web Services** lanzó la primera cloud pública, con los servicios Elastic Compute Cloud (EC2) y Simple Storage Service (S3). El primero ofrece cómputo virtual y el segundo almacenamiento de datos en cubos o *buckets*. De ahí en adelante surgieron más servicios como **Google Cloud** y **Microsoft Azure** en 2008.

Desde la década pasada hasta al punto actual se han popularizado muchos servicios basados en la nube. Dropbox o Google Drive para almacenar archivos, Office 365 como herramienta para gestionar la actividad empresarial, herramientas del sector público y últimamente incluso aplicaciones *cloud native*, que están desplegadas en la nube en su totalidad.

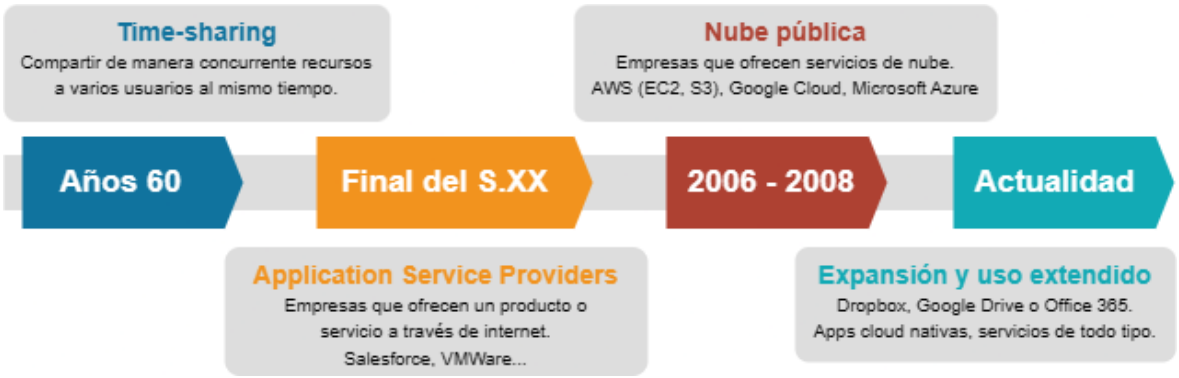



**Figura 3.3:** *Linea de tiempo de la historia del Cloud Computing. Fuente: elaboración propia.*

### 3.2.2. Tendencias y cifras del mercado

Para entender la magnitud de esta tecnología, es importante hablar sobre cifras e impacto económico del sector y para ello se usará el análisis hecho por una institución reputada. Según el informe [50] realizado por la empresa International Data Corporation (IDC), el gasto mundial en servicios en la nube podría llegar a ser de unos **805 mil millones de dólares en 2024**, con una tasa de crecimiento anual compuesto del 19.4 % hasta 2028.

En base al análisis de la consultora inglesa, Norteamérica lidera el gasto con 432 mil millones, Europa desembolsará 167 mil millones, entre los que destaca el **sector de banca y retail** como mayor consumidor. Aún así, China no se queda atrás, ya que **Alibaba** ha anunciado recientemente una inversión multimillonaria (52 mil millones) en Inteligencia Artificial y Cloud según reporta la Agencia EFE [51].

Hay que diferenciar entre los distintos tipos de modelos de computación en la nube y para ello es útil visualizar el gasto proporcional con un gráfico (figura 3.4) y una explicación sobre qué significa cada uno:

- **SaaS** (Software as a Service).
  El usuario accede a aplicaciones completas a través de internet, sin tenerse que preocupar por su instalación o mantenimiento (Gmail, Salesforce o ChatGPT).
- **PaaS** (Platform as a Service).
  Se proporciona un entorno listo para desarrollar, probar y desplegar aplicaciones sin gestionar la infraestructura (Google App Engine, Heroku o AWS Elastic Beanstalk).
- **IaaS** (Infrastructure as a Service).
  Se ofrecen recursos básicos de computación como servidores, almacenamiento o redes en la nube y el usuario tiene que configurar y administrar todo lo relacionado (AWS EC2, Azure VMs o Google Compute Engine).

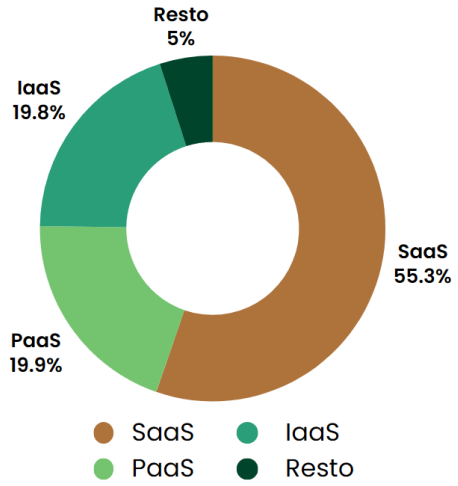



**Figura 3.4:** *Porcentaje de gasto de cada tipo de computación en la nube. Fuente: elaboración propia.*

Durante toda esta sección hemos mencionado varias empresas que se encargan de proveer este tipo de tecnología, se ha realizado una comparativa con datos a mayo de 2025 (tabla 3.3) entre las más importantes para decidir cual es la más adecuada en el caso de este trabajo.

Los proveedores ordenados según su cuota de mercado (datos de [52]) son estos:

1. **Amazon Web Services** (véase 3.2.3). AWS es el mayor proveedor del mundo, líder en el sector, con una cuota de mercado de más del 30 % en el Q4 de 2024. Tiene unos precios muy competitivos, una infraestructura sólida alrededor del mundo y una gama de servicios muy amplia.
2. **Microsoft Azure**. Con cerca de un 21 % de la porción de mercado, Azure es el segundo en la lista, y esto se debe a que está integrado con los productos de Microsoft, además de sus herramientas para *big data* e inteligencia artificial.
3. **Google Cloud Platform** (GCP). Ocupa el tercer puesto con aproximadamente un 12 % del mercado, principalmente por sus servicios de análisis avanzado de datos, despliegue con contenedores y aprendizaje automático.
4. **Alibaba Cloud**. Con alrededor de un 4 % de cuota, es el principal proveedor en el continente asiático, especialmente en China. Aunque apenas tiene presencia en Occidente, está creciendo rápidamente en mercados emergentes.
5. **Otros**. El resto del mercado (alrededor del 23 %) está repartido entre proveedores más pequeños como IBM Cloud, Oracle Cloud, Salesforce, VMware, Tencent Cloud... Estos ofrecen servicios más especializados o enfocados a mercados concretos.

Para elegir entre ellos, se han tenido en cuenta ciertas consideraciones entre los 3 proveedores que tienen una región o zona de disponibilidad en España, para tener menor latencia y sobre todo cumplir las leyes y la Ley de Protección de Datos (RGPD).

| **Criterio** | **AWS** | **Azure** | **GCP** | **Factor** |
|---|---|---|---|---|
| Capa gratuita | 3 | 3 | 4 | 0.2 |
| Cantidad de servicios | 5 | 4 | 3 | 0.1 |
| Conocimiento del autor | 4 | 1 | 2 | 0.2 |
| Integración | 3 | 5 | 4 | 0.1 |
| Comunidad y soporte | 5 | 4 | 3 | 0.2 |
| Motivación personal | 5 | 4 | 3 | 0.2 |
| **Resultado** | **4.2** | **3.3** | **3.1** | |

**Tabla 3.3:** *Evaluación ponderada de proveedores cloud. Fuente: elaboración propia.*

En consecuencia a lo analizado en esta sección, se ha elegido AWS como proveedor de nube para este trabajo [15], y se explicarán en detalle los servicios utilizados y el funcionamiento general del mismo.

[15] Aunque la infraestructura principal se ha desplegado en AWS, se ha utilizado también **Firebase** exclusivamente para la autenticación de usuarios, aprovechando su facilidad de integración.

### 3.2.3. Amazon Web Services

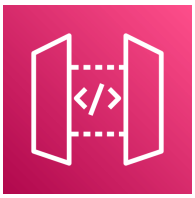
En esta sección se explicarán brevemente, junto con sus iconos respectivos para ayudar a identificarlos en los diagramas de arquitectura futuros, ciertos servicios del sistema creado por Amazon (para más detalle: [53]), los que tienen una influencia directa en este proyecto principalmente. Aún así, para tener una visión general de todos los productos que ofrece AWS, es muy recomendable ver este vídeo [16] de Fireship, que resume los servicios más utilizados en la industria.

**Lambda** [54] permite ejecutar código (en muchos lenguajes) de manera *serverless* o sin servidor, que quiere decir que no hay que ocuparse de gestionar servidores, sino que AWS los provisiona directamente y se encarga del mantenimiento. De esta manera, es mucho más sencillo tener un backend funcional, solo se paga por el tiempo de ejecución y se puede conectar de manera sencilla a otros servicios.

**S3** [55] es un servicio de almacenamiento de objetos en cubos o *buckets*. Permite guardar archivos de cualquier tipo y tamaño, y acceder a ellos bajo demanda. Hay varios tipos de almacenamientos:

- *Standard*, acceso inmediato y velocidad de descarga directa.
- *Glacier*, pensado para archivos con acceso infrecuente como copias de seguridad, a un precio reducido.
- *One-Zone*, solo en una zona de disponibilidad [17], a menor coste pero misma velocidad que *Standard*.

**DynamoDB** [56] es una base de datos NoSQL [18] gestionada por AWS, similar a MongoDB, para la que no hace falta tener servidores, sino que funciona de manera *serverless*. Esta permite flexibilidad en los objetos que se guardan, y es muy útil para guardar los eventos de este trabajo.



**API Gateway** [57] es el servicio que permite crear, publicar y mantener APIs REST, HTTP o WebSocket. Estas actúan como puertas de entrada para que tanto aplicaciones externas como servicios de AWS puedan acceder o modificar datos por ejemplo, de una base de datos.

[16] Top 50+ AWS Services Explained in 10 Minutes. Vídeo.

[17] Una zona de disponibilidad **es el conjunto de uno o más centros de datos en una zona geográfica concreta**. No hay que confundirlo con región, ya que en una región puede haber más de una zona de disponibilidad.

[18] Una base de datos NoSQL no es relacional, sino que usa un sistema de clave-valor o de documentos para almacenar la información.

### Casos de uso

La instancia más famosa sobre el uso de AWS es el caso de **Netflix** [58], utiliza AWS para escalar globalmente su infraestructura, garantizando disponibilidad, rendimiento y almacenamiento de alto volumen para millones de usuarios al mismo tiempo. Esto lo hace utilizando S3 para almacenaje, EC2 o *Elastic Compute Cloud* para servidores, Lambda para computación y otros servicios distintos.

Un análisis de varios casos de uso sobre la inteligencia artificial en la plataforma de Amazon es el que realizan Brian Kan y Douglas Klein en su artículo [59], del que hay que destacar empresas como **Disney**, a la que AWS ayudó a migrar sus servidores a la nube y usar modelos de aprendizaje automático, **T-Mobile** con la implementación de chatbots y reconocimiento de voz, o la propia **NFL**, que usó **SageMaker** (una plataforma de integración de herramientas de Machine Learning e IA) para analizar estadísticas de los jugadores en el campo a tiempo real.

### Costes y capa gratuita

A continuación, en la tabla 3.4 se describen los costes aproximados de estos cuatro servicios y las capas gratuitas que Amazon ofrece (algunas durante los primeros 12 meses y otras indefinidamente, a fecha de mayo de 2025), que se han aprovechado para no tener gastos de computación durante el proyecto.

| | **Lambda** | **S3** | **DynamoDB** | **API Gateway** |
|---|---|---|---|---|
| **Capa gratuita** | 1 millón de invocaciones o 400.000 GB/s | 5 GB de almacenamiento en clase *Standard* | 25 GB de almacenamiento 200M lecturas y 2.5M escrituras | 1 millón de llamadas REST al mes |
| **Coste adicional** | \$0.20 por millón de invocaciones | Desde \$0.023 por GB/mes [19] | \$0.25 por WCU* \$0.0065 por RCU* | \$3.50 por millón de llamadas REST |

**Tabla 3.4:** *Capa gratuita y costes adicionales de los servicios de AWS. Fuente: [60] Un WCU* equivale a 1KB/s de escritura y un RCU* a 4KB/s de lectura.*

Una vez analizados los costes, se llega a la conclusión de que es relativamente sencillo no pasarse de la capa gratuita para un proyecto piloto como lo es AgroFind. De todos modos, el objetivo principal del proyecto es la trazabilidad, y hay una tecnología diseñada con ese propósito que es la *blockchain*. En la siguiente sección se explicará qué es, su historia y antecedentes de manera breve, y se decidirá que cadena elegir para este trabajo.

[19] Precio aproximado en región eu-south-2 (Madrid) según AWS Calculator. Puede variar en otras regiones.

## 3.3. Blockchain

Según el glosario del NIST [61], una blockchain o cadena de bloques (en inglés y castellano) es:

> A distributed digital ledger of cryptographically-signed transactions that are grouped into blocks. Each block is cryptographically linked to the previous one (making it tamper evident) after validation and undergoing a consensus decision.
>
> ---
>
> Un libro de contabilidad digital y distribuido que registra transacciones firmadas criptográficamente y estas se agrupan en bloques. Cada bloque está unido criptográficamente al anterior (haciendo que sea a prueba de manipulaciones) después de haber pasado una validación y una decisión de consenso.

En esta definición hay varios conceptos que hay que aclarar. Primeramente, un libro de contabilidad (*ledger*) distribuido se refiere al registro de transacciones del que hay múltiples copias del mismo en lugares distintos, esto hace que en caso de algún ataque sea necesario tomar el control de más de la mitad de ellos [20].

Siguiendo, el emisor de la transacción firma siempre la misma utilizando un **cifrado asimétrico**, que consiste en generar un par de claves generadas por algún algoritmo, normalmente el Algoritmo de Firma Digital de Curva Elíptica o ECDSA. De este algoritmo [62] se obtiene una clave pública y una privada. La clave privada se usa para firmar la transacción y solo es conocida por el autor de la misma, y la clave pública se usa para verificar que la firma es auténtica y que la transacción no ha sido modificada de ninguna manera desde que se firmó.

Para entender los bloques, la validación y el consenso es mejor recurrir al origen de la primera cadena de bloques, que fue **Bitcoin**.

### 3.3.1. Bitcoin

Como describe el paper o whitepaper original de Satoshi Nakamoto [21] llamado *Bitcoin: A Peer-to-Peer Electronic Cash System* [63], cada bloque contiene un encabezado y una lista de transacciones. El encabezado incluye un ***hash*** del bloque anterior. Este *hash* es una huella digital criptográfica única del bloque anterior. Esta interconexión crea una cadena inquebrantable: cualquier intento de modificar un bloque anterior invalidaría todos los bloques siguientes, ya que sus *hashes* no coincidirían. Esto es lo que se define como **inmutabilidad**.

[20] A esto se le llama ataque del 51 %, y es la mayor debilidad de los sistemas distribuidos a la vez que su mayor fortaleza, porque realizar este tipo de ataques a blockchains a gran escala es muy poco probable. Fuente: https://www.coinbase.com/es-es/learn/crypto-glossary/what-is-a-51-percent-attack-and-what-are-the-risks

[21] De hecho, Satoshi Nakamoto es un pseudónimo [64], y aún a día de hoy no se sabe si quién ideó Bitcoin fue un individuo o un grupo de personas.

Para añadir un nuevo bloque a la cadena, los nodos de la red (conocidos como *mineros*) compiten para resolver un problema computacional complejo, conocido como **Proof of Work** o PoW, que consiste en encontrar un *nonce* que dada la cabecera del bloque, incluyendo este propio *nonce*, cree un hash que sea menor a la dificultad de minado en ese momento en el tiempo. El primero en encontrar la solución puede añadir el nuevo bloque y obtiene como recompensa una cantidad determinada de Bitcoin, que es la criptomoneda asociada a esta blockchain.

La blockchain no es controlada por una única entidad, por eso se dice que es distribuida, porque cada nodo de la red tiene una copia completa o parcial de la cadena de bloques, el conocido como *ledger*. Cuando se propone un nuevo bloque, los nodos lo validan (véase [65], sección II, b) de forma independiente para asegurarse de que las transacciones sean legítimas y que el bloque siga las reglas del protocolo. Para una explicación más matemática es recomendable leer *Rewriting History in Bitcoin and Friends* [66], en el que hacen un análisis profundo sobre la complejidad detrás de este sistema haciendo uno desde cero. Es interesante comentar que el ataque del 51 % mencionado previamente se conoce como el problema de los generales bizantinos, analizado en [67].

En la teoría Bitcoin se diseñó como un sistema de dinero electrónico o reserva de valor (por mucho que ahora se esté usando su moneda para especular y no con una utilidad directa), pero no está pensada para almacenar otra cosa que no sean transacciones económicas. Con la idea de crear aplicaciones descentralizadas (DApps), tener una plataforma más versátil y a menor coste nace Ethereum.

### 3.3.2. Ethereum

Ethereum es una plataforma de blockchain descentralizada y de código abierto [22] que Vitalik Buterin formuló en su famoso artículo [23] *A next generation smart contract and decentralized application platform.* [68] en 2013 y que se puso en marcha en 2014.

Como explica la propia web de Ethereum [69] y aparece en el whitepaper de Buterin, lo revolucionario de esta tecnología es que la propia blockchain posee un lenguaje de programación Turing completo, y por tanto esto permite a los desarrolladores construir sus aplicaciones de manera mucho mas sencilla, usando los ***smart contracts*** o **contratos inteligentes**. Estos son programas autoejecutables almacenados en la blockchain. Las reglas del contrato están programadas y una vez que se cumplen las condiciones predefinidas, se ejecuta automáticamente sin necesidad de ningún intermediario. Esto permite la creación de acuerdos automatizados y de confianza.

Para poder hacer funcionar todo el sistema, en esta red existe un entorno de ejecución muy similar al explicado en el capítulo 3.1, que es la EVM. Esta puede ejecutar cualquier cálculo que un ordenador normal podría realizar y por tanto es muy flexible para los desarrolladores.

[22] Página de Github con los clientes e implementaciones: https://github.com/ethereum

[23] Un whitepaper es artículo/documento que explica algunos conceptos en detalle y que, generalmente, no ha pasado por un proceso de revisión como hay en revistas científicas o congresos.

Cualquier contrato inteligente que se quiera hacer en la blockchain se programa usando **Solidity** [70], que tiene su propia herramienta de desarrollo: **Remix**[24], un entorno de desarrollo integrado (IDE) nativo de navegador. El lenguaje más similar a Solidity sería JavaScript, por ser fuertemente tipado, orientado a objetos y por compartir una sintaxis similar, aunque está especialmente diseñado para la programación de contratos inteligentes y presenta particularidades en cuanto a gestión de memoria y tiene ciertas funciones especiales como `payable` o `view`. Un extracto de código de ejemplo se expone a continuación (parte del contrato de AgroFind):

```
pragma solidity ^0.8.29; // se usa para definir versiones
contract RegistroEventosParcela {
    event EventoRegistrado(
        bytes32 indexed hashEvento,
        bytes32 indexed hashParcela,
        uint256 timestamp
    ); /* resto de contrato */
```

**Extracto de código 3.4:** *Parte del contrato RegistroEventosParcela.*

Ethereum inició usando PoW para el funcionamiento de los nodos y las validaciones al igual que Bitcoin, pero para intentar mejorar la escalabilidad, en septiembre de 2022 se realizó la transición a ***Proof of Stake***. En el PoS (véase 3.5), los validadores se seleccionan para crear nuevos bloques en base a la cantidad de Ether (ETH, la criptomoneda nativa de Ethereum) que han depositado como garantía.

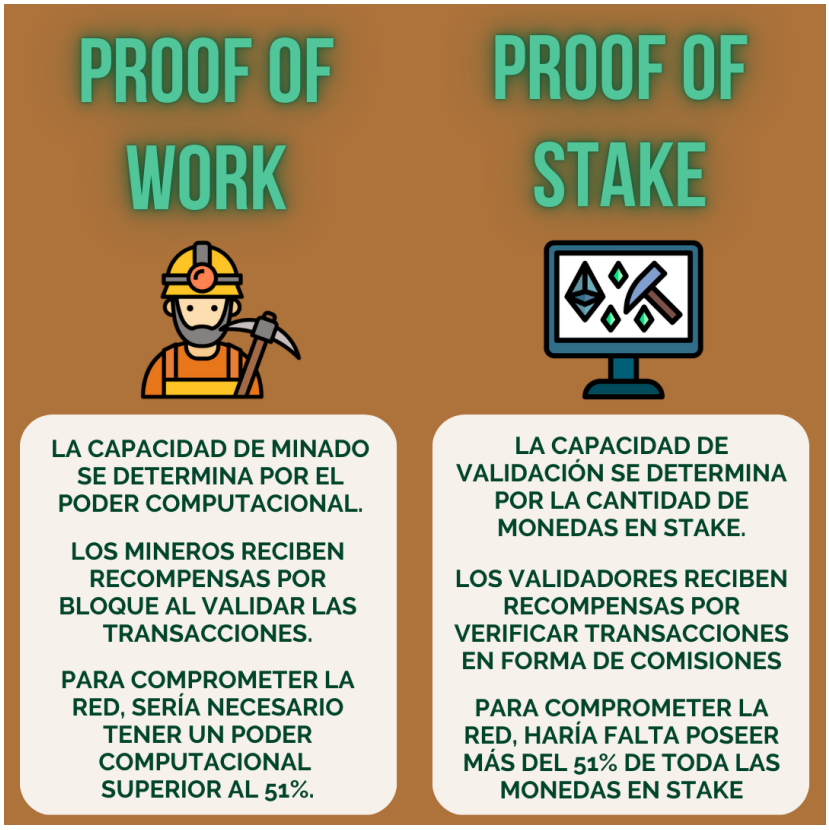



**Figura 3.5:** *PoW vs PoS. Fuente: elaboración propia usando* https://www.canva.com/.

[24]Enlace a la IDE de Remix: https://remix.ethereum.org/

Haber cambiado a PoS conlleva que el impacto computacional sea mucho menor que en PoW, y por tanto lo que hace que un validador tenga más poder de votación es la cantidad de ETH que tiene depositado, que no se puede sacar hasta que no finalice un periodo determinado de tiempo [25].

El Ether se utiliza para pagar el coste computacional para ejecutar transacciones y smart contracts en la red. A este coste se le suele llamar **gas** o **tarifa de gas**, y su medida es el Gwei, donde 1 Gwei = $10^{-9}$ Ether. Este gas tiene un coste monetario real, según datos de Ethereum Gas Tracker [26] e YCharts [27] el precio medio por transacción fluctúa entre 0.87$ en las más baratas hasta 70.49$ en las más caras, a mayo de 2025.

El problema principal de Ethereum siguen siendo los costes de gas, ya que a un inversor o especulador que simplemente mueva dinero no le afecta tanto, pero un desarrollador que tenga ganas de hacer un juego que requiera varias transacciones al día sale muy perjudicado, por el coste tan elevado que tienen las mismas debido a las comisiones que se llevan los validadores. Para solucionar esto y hacer las transacciones más rápidas, no necesariamente en Ethereum sino en Bitcoin (mediante Lightning Network [71]) nace el concepto de *layer 2*, que es básicamente otra blockchain que funciona simultáneamente a la *layer 1* o principal, en la que al hacerse un número grande de transacciones, se empaquetan (*rollup*) y se publican como un bloque en la blockchain principal con pruebas de que esas transacciones son válidas.

Para elegir una blockchain para realizar el proyecto, se hace una comparativa similar a las anteriores, pero esta vez con datos de una tesis doctoral hecha por Laurin Zubler [72], porque hace un excelente trabajo en recopilar la información de manera relativamente reciente (octubre de 2024), actualizados a hoy en día, con el precio del gas y reemplazando dos de ellos:

| Abrev. | Ranking | Nodos | Lenguaje | Velocidad | Precio por tx. |
|---|---|---|---|---|---|
| ETH | 2 | PoS | Solidity | hasta 5mins | 1 - 70+$ |
| BNB | 5 | PoSA | Solidity | 3 - 5s | 0.05 - 0.30$ |
| SOL | 6 | PoS | Rust | instantáneo | $<$ 0.01$ |
| TRX | 10 | DPoS | Solidity | 1 - 3s | $<$ 0.01$ |
| SUI | 11 | DPoS | Move | instantáneo | $<$ 0.01$ |
| POL | 45 | PoS | Solidity | 1 - 2s | 0.005 - 0.02$ |
| ARB | 49 | Rollup | Solidity | instantáneo | 0.01 - 0.10$ |
| OP | 65 | Rollup | Solidity | 1s | 0.01 - 0.10$ |

**Tabla 3.5:** *Comparativa de diferentes blockchains y sus características.*

[25] Esto se conoce como *staking*, y es una de las maneras mas comunes de sacar rentabilidad sin especular sobre el valor de la criptomoneda en cuestión.

[26] Consulta de gas a tiempo real, enlace: https://etherscan.io/gastracker

[27] Precio medio de Gas en ETH, enlace: https://ycharts.com/indicators/ethereum_average_gas_price

| | Precio | Velocidad | Comunidad | Pref. Personal | Resultado |
|---|---|---|---|---|---|
| Factor | 0.4 | 0.2 | 0.2 | 0.2 | |
| Ethereum | 1 | 1 | 5 | 5 | 2.6 |
| **Solana** | 5 | 5 | 4 | 2 | **4.2** |
| BSC | 3 | 3 | 4 | 5 | 3.6 |
| TRON | 5 | 4 | 2 | 2 | 3.6 |
| Sui | 5 | 5 | 3 | 2 | 4 |
| **Polygon** | 4 | 4 | 4 | 5 | **4.2** |
| Arbitrum | 4 | 5 | 3 | 3 | 3.8 |
| Optimism | 4 | 5 | 3 | 3 | 3.8 |

**Tabla 3.6:** *Evaluación ponderada de blockchains a seleccionar.*

Entre las dos blockchains con mejor puntuación, se ha decidido optar por **Polygon**, porque tiene una red de pruebas más sencilla y porque **Solana** se programa en *Rust*, con más complejidad para un contrato simple como el que se usará en este caso. La única pega que tiene Polygon en la red de prueba es cómo conseguir el saldo (las conocidas como *faucets*), que se ha conseguido solventar de manera sencilla.

### 3.3.3. Matic / Polygon

Matic Network originalmente, ahora llamado Polygon [28] fue originalmente una solución de escalado de Layer 2 para Ethereum, mediante la construcción de una cadena de bloques alternativa, llamada *sidechain* que es compatible con la EVM.

Matic utilizaba una arquitectura PoS para su sidechain [73] y los usuarios podían mover sus activos entre Ethereum y la cadena de Matic utilizando puentes [29] o *bridges*, pero para poder abarcar mejor el mercado, se hizo una migración, en la que todo el ecosistema pasó a llamarse **Polygon**.

#### Migración a Polygon [74]

En febrero de 2021, Polygon se posicionó como un ecosistema de escalado modular y flexible, creando un *internet de blockchains*, para que diferentes cadenas puedan interactuar y comunicarse entre sí de manera eficiente. De esta manera, lo que era antes Matic, pasó a ser **Polygon PoS Chain** [75], que es la blockchain que se usará para este trabajo. Polygon ahora ofrece más herramientas tanto una blockchain zkEVM [30] cómo otras tantas que no se analizarán en este trabajo.

[28] Página web oficial de Polygon, enlace: https://polygon.technology/
[29] Los puentes funcionan bloqueando activos en una cadena de origen a través de un smart contract y emitiendo activos equivalentes en una cadena de destino.
[30] Polygon zkEVM, enlace: https://polygon.technology/polygon-zkevm

Para no tener que incurrir en ningún coste, se ha hecho todo el contrato inteligente en la red de pruebas **Polygon Amoy** [76], de la que se puede sin ningún problema desplegar a la red principal, debido a que el contrato en Solidity tiene los mismos parámetros exactamente. La única diferencia es el coste, porque se explicará en la arquitectura que se usa un **relayer**[31], y habría que introducir fondos al mismo para hacer las transacciones.

Es importante denotar que se planteó usar una blockchain creada desde cero, pero el beneficio de tener algo descentralizado y que se pueda consultar fácilmente al igual que PolygonScan [32] (cómo se hará mediante códigos QR), que ofrece una web para visualizar las transacciones, los smart contracts y los eventos de los mismos.

Como se ha comentado antes, al hacer el smart contract en una red de pruebas, hay que tener una manera fiable de obtener saldo de prueba para la(s) billetera(s) que hará(n) estas transacciones. Por ello, se han empleado dos faucets [33]. La primera ha sido la que ofrece Alchemy (fiable, en pequeñas cantidades), https://www.alchemy.com/faucets/polygon-amoy y la segunda ha sido la que ofrece Stakepool (más cantidad pero menos fiable), https://faucet.stakepool.dev.br/amoy.

[31] Explicación más detallada sobre que es un *relayer* en el capítulo 6 y porqué es lo que hace que la aplicación sea accesible a los agricultores.

[32] Enlace al explorador de bloques de Polygon: https://polygonscan.com/

[33] Una faucet (grifo) es una página web que distribuye pequeñas cantidades de criptomonedas de forma gratuita, normalmente en redes de prueba, para que los usuarios puedan experimentar con la tecnología o realizar pruebas.

# 4. Planificación

## 4.1. Herramientas

A continuación se detallan las herramientas utilizadas para el desarrollo del proyecto, ordenadas por secciones en base a para qué se han utilizado, acompañadas de una breve descripción y un enlace a sus páginas web.

### Diseño y documentación

- **Gimp**[34]**:** Software de código abierto de edición de imágenes utilizado para retocar elementos gráficos, tanto para la interfaz de usuario como para el logo.
- **Canva**[35]**:** Herramienta de diseño gráfico online empleada para elaborar materiales visuales varios, dentro de la aplicación y en la documentación (ej. 3.3, 3.5).
- **Draw.io**[36]**:** Aplicación de diagramación utilizada para diseñar la arquitectura cloud, la arquitectura final y diagramas como el EDT (4.1).
- **GanttPRO**[37]**:** Software de gestión de proyectos para planificar y visualizar las tareas del proyecto mediante diagramas de Gantt (4.2, 4.3, 4.4, 4.5).
- **Overleaf**[38]**:** Editor de LaTeX online, utilizado para redactar de la memoria.
- **Google Scholar**[39]**:** Motor de búsqueda de literatura académica, usado para investigar artículos y publicaciones relevantes para el proyecto.
- **PlantUML**[40]**:** Software para crear diagramas UML y esquemas varios.

### Colaboración y gestión de archivos

- **Google Drive**[41]**:** Servicio de almacenamiento en la nube utilizado para almacenar documentos del proyecto y copias de seguridad del proyecto y de la memoria.
- **Outlook**[42]**:** Usado para comunicación con el tutor del proyecto.

[34]Enlace: http://www.gimp.org.es/
[35]Enlace: https://www.canva.com/
[36]Enlace: https://app.diagrams.net/
[37]Enlace: https://ganttpro.com/
[38]Enlace: https://www.overleaf.com/
[39]Enlace: https://scholar.google.com/
[40]Enlace: https://plantuml.com/es/
[41]Enlace: https://www.google.com/drive/
[42]Enlace: https://outlook.live.com/

- **GitHub**[43]**:** Plataforma de control de versiones basada en Git, esencial para gestionar el código fuente del proyecto y poder trabajar desde distintos dispositivos.
- **Brave**[44]**:** Navegador web usado para muchas tareas.

## Blockchain, desarrollo y pruebas

- **Remix**[45]**:** IDE basado en navegador para escribir, compilar y desplegar los contratos inteligentes de Solidity.
- **Metamask**[46]**:** Billetera de criptomonedas (extensión de navegador) usada para interactuar con la red blockchain de Polygon para probar y gestionar las transacciones, además de para añadir saldo de prueba al *relayer*.
- **Polygon Scanner**[47]**:** Explorador de bloques para la red Polygon, utilizado para comprobar que el smart contract esté funcionando correctamente.
- **Faucets:** Servicios que han proporcionado pequeñas cantidades de tokens para las pruebas en la red de desarrollo de Polygon, citadas anteriormente.
- **Flutter SDK**[48]**:** Kit de desarrollo de software de Google utilizado para construir la interfaz de usuario para la aplicación de AgroFind.
- **Teléfono móvil:** Dispositivo físico, más concretamente un *Xiaomi Mi A2*, usado para probar la aplicación en un entorno real y verificarla.
- **Emulador Android:** Software que simula un dispositivo Android en el ordenador, permitiendo probar la aplicación en otro entorno distinto.
- **Visor SIGPAC**[49]**:** Herramienta utilizada para visualizar información geográfica de parcelas agrícolas.
- **Visual Studio Code**[50]**:** IDE utilizado para la codificación general del proyecto, incluyendo el desarrollo móvil con Flutter, las *lambdas* en Python y en NodeJS.

## Servicios en la nube y APIs

- **Firebase**[51]**:** Plataforma de desarrollo de aplicaciones de Google, usada para la autenticación de usuarios y para ejecutar pruebas usando **Robo Test**.

[43]Enlace: https://github.com/
[44]Enlace: https://brave.com/
[45]Enlace: https://remix.ethereum.org/
[46]Enlace: https://metamask.io/
[47]Enlace: https://polygonscan.com/
[48]Enlace: https://flutter.dev/
[49]Enlace: https://sigpac.mapama.gob.es/fega/visor/
[50]Enlace: https://code.visualstudio.com/
[51]Enlace: https://firebase.google.com/

- **Consola de AWS**[52]**:** Consola web que permite gestionar todo lo relacionado con Amazon Web Services en este proyecto (la API, Lambdas, DynamoDB...).
- **AWS Cost Explorer**[53]**:** Herramienta de AWS para consultar todo lo relacionado con la capa gratuita y hacer cálculos para no sobrepasar esta.
- **AWS CLI**[54]**:** Interfaz de línea de comandos para interactuar con los servicios de AWS desde la terminal.
- **Postman**[55]**:** Herramienta para el desarrollo de APIs, utilizada para probar y documentar la API.
- **WeatherAPI**[56]**:** API gratuita de previsión de tiempo.
- **SIGPAC**[57]**:** Varias APIs que ofrece el MAPA con información en torno a SIGPAC: detalles de parcelas, tipos de uso, listas de códigos SIGPAC...

### IA y asistencia al desarrollo

- **LeonardoAI**[58]**:** Plataforma de generación de imágenes por IA, utilizada para crear la base del logo final.
- **ChatGPT**[59]**:** Modelo de lenguaje, empleado para la ayuda en la documentación.
- **GitHub Copilot**[60]**:** Asistente de codificación, utilizado para acelerar la programación, siempre en partes no-críticas del proyecto (interfaces principalmente).

Aparte de las herramientas, también es importante mencionar que tecnologías de programación y bases de datos se han empleado:
**Python** para el backend en AWS, **NodeJS** para interactuar con la blockchain, **Dart y Flutter** para el desarrollo de la interfaz de usuario y la lógica de la aplicación, **Solidity** para escribir los contratos inteligentes de Polygon, **pub** para gestionar las librerías que la aplicación utiliza, **DynamoDB** para almacenar datos no estructurados en la nube de AWS y **SQFlite**, para el almacenamiento local de datos en la aplicación.

## 4.2. Alcance

En esta sección se incluye el diagrama de Estructura de Descomposición del Trabajo, y se explican los bloques principales del mismo. Para facilitar su visualización, se ha girado la siguiente página donde está el diagrama (figura 4.1). Además, se detalla el listado de las tareas.

[52]Enlace: https://aws.amazon.com/console/
[53]Enlace: https://aws.amazon.com/es/aws-cost-management/aws-cost-explorer/
[54]Enlace: https://aws.amazon.com/cli/
[55]Enlace: https://www.postman.com/
[56]Enlace: https://www.weatherapi.com/
[57]Enlace: https://sigpac-hubcloud.es/
[58]Enlace: https://leonardo.ai/
[59]Enlace: https://chat.openai.com/
[60]Enlace: https://github.com/features/copilot

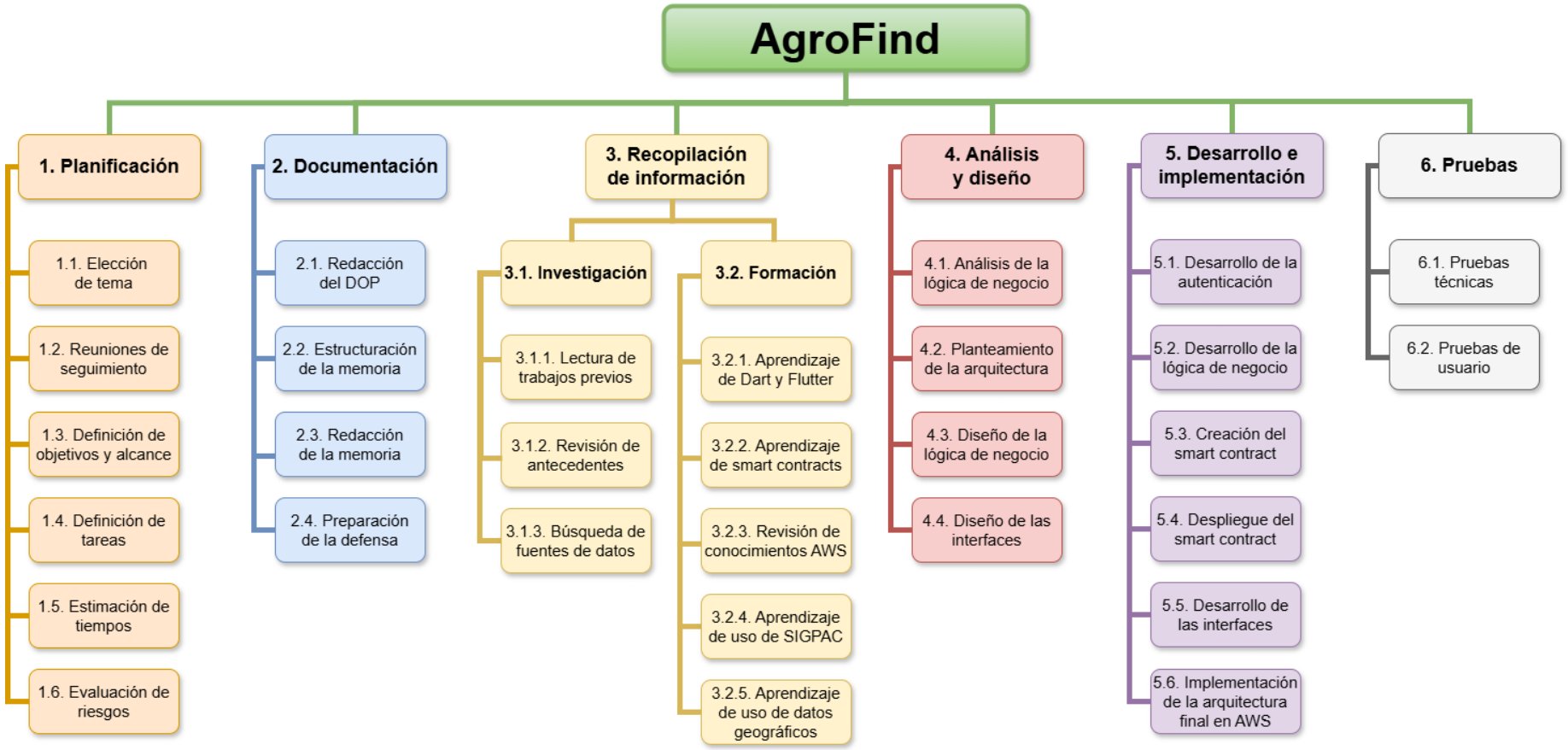



**Figura 4.1:** *Estructura de desglose del trabajo (EDT) de AgroFind. Hecho con* draw.io.

### 4.2.1. Listado de tareas

Se definirán todas las tareas, cada una en su sección y para ello se usará una tabla de referencia como la siguiente:

| **A.B.C. Nombre de la tarea.** | |
|---|---|
| Descripción | Descripción breve de la tarea. |
| Duración (h) | Duración estimada en horas. |
| Recursos necesarios | Lista de herramientas necesarias. |
| Precedente | Tareas necesarias para comenzar esta. |
| Salidas | Explicación de los hitos obtenidos (si procede). |

**Tabla 4.1:** *Ejemplo de detalle de una tarea.*

1. **Planificación**. La fase inicial de cualquier proyecto, que comenzó desde el primer *brainstorming* para idear un tema. Se centra en establecer las bases para el desarrollo e incluye definir tanto el alcance del proyecto como los objetivos y paquetes de trabajo a completar. También engloba la estimación de tiempos y la anticipación de posibles riesgos, además de las reuniones de seguimiento con el tutor.

| **1.1. Elección de tema** | |
|---|---|
| Descripción | Definir el área del proyecto y que solución específica a un problema se plantea. |
| Duración (h) | Estimada: 12 horas. |
| Recursos necesarios | Acceso a internet para lectura de otros trabajos, tablero de corcho para pegar las ideas en el *brainstorming*. |
| Precedente | Ninguno. |
| Salidas | Tema de proyecto definido y un borrador inicial de objetivos del proyecto. |

**Tabla 4.2:** *Descripción de tarea: Elección de tema.*

| **1.2. Reuniones de seguimiento** | |
|---|---|
| Descripción | Primera reunión con el tutor y el resto de ellas a lo largo del proyecto. Se informará sobre los progresos y consulta de dudas, además de hitos a cumplir. |
| Duración (h) | 2 horas por reunión, total 12 horas. |
| Recursos necesarios | Outlook para intercambiar correos. |
| Precedente | 1.1. (véase 4.2). |
| Salidas | Acuerdos sobre la dinámica de las reuniones, y siguiente reunión concretada. |

**Tabla 4.3:** *Descripción de tarea: Reuniones de seguimiento.*

| **1.3. Definición de objetivos y alcance** | |
|---|---|
| Descripción | Aclarar qué se espera lograr con el proyecto y cuales son los límites del mismo. |
| Duración (h) | Estimada: 6 horas. |
| Recursos necesarios | Papel y bolígrafo. |
| Precedente | 1.1. (véase 4.2). |
| Salidas | Estructura inicial del documento de objetivos del proyecto (DOP). |

**Tabla 4.4:** *Descripción de tarea: Definición de objetivos y alcance.*

| **1.4. Definición de tareas** | |
|---|---|
| Descripción | Desglosar el proyecto en paquetes de trabajo más pequeños y cuantificables, identificando las dependencias entre ellos. |
| Duración (h) | Estimada: 8 horas. |
| Recursos necesarios | Estructura inicial del DOP y un dispositivo con acceso a internet. |
| Precedente | 1.3. (véase 4.4). |
| Salidas | Listado de las tareas y Estructura del Desglose del Trabajo (EDT). |

**Tabla 4.5:** *Descripción de tarea: Definición de tareas.*

| **1.5. Estimación de tiempos** | |
|---|---|
| Descripción | Asignar una duración estimada a cada tarea definida en el EDT. |
| Duración (h) | Estimada: 4 horas. |
| Recursos necesarios | EDT de 1.4. y datos de otros proyectos similares. |
| Precedente | 1.4. (véase 4.5). |
| Salidas | Diagrama de Gantt preliminar e identificación de hitos. |

**Tabla 4.6:** *Descripción de tarea: Estimación de tiempos.*

| **1.6. Evaluación de riesgos** | |
|---|---|
| Descripción | Identificar los posibles problemas o amenazas que puedan afectar el proyecto (en tiempo, coste, alcance o calidad) y hacer estrategias de mitigación y planes de contingencia para cada uno. |
| Duración (h) | Estimada: 4 horas. |
| Recursos necesarios | Lista de tareas de 1.4. |
| Precedente | 1.4. (véase 4.5). |
| Salidas | Registro de posibles riesgos del proyecto, planes de mitigación y contingencia. |

**Tabla 4.7:** *Descripción de tarea: Evaluación de riesgos.*

2. **Documentación**. Incluye la redacción del Documento de Objetivos del Proyecto (DOP), la estructura y escritura de la memoria y gestionar la defensa del trabajo.

| **2.1. Redacción del DOP** | |
|---|---|
| Descripción | Elaborar el Documento de Objetivos del Proyecto (DOP). |
| Duración (h) | Estimada: 16 horas. |
| Recursos necesarios | Estructura inicial del DOP y Overleaf. |
| Precedente | 1.6. (véase 4.7). |
| Salidas | Documento de Operación del Proyecto (DOP) finalizado. |

**Tabla 4.8:** *Descripción de tarea: Redacción del DOP.*

| **2.2. Estructuración de la memoria** | |
|---|---|
| Descripción | Organizar el índice y la estructura de la memoria final del proyecto. |
| Duración (h) | Estimada: 4 horas. |
| Recursos necesarios | Otros proyectos similares, plantilla de LaTeX de otra asignatura hecha por el autor y una reunión con el tutor. |
| Precedente | 2.1 (véase 4.8). |
| Salidas | Índice de la memoria del proyecto y plantilla de la misma definida. |

**Tabla 4.9:** *Descripción de tarea: Estructuración de la memoria.*

| **2.3. Redacción de la memoria** | |
|---|---|
| Descripción | Escribir el contenido completo de la memoria del proyecto, incluyendo todo lo necesario. |
| Duración (h) | Estimada: 80 horas. |
| Recursos necesarios | Todo el proyecto, revisión del tutor. |
| Precedente | 2.2., 5. al completo y 6. al 50 %. |
| Salidas | Memoria completa, revisada por el tutor. |

**Tabla 4.10:** *Descripción de tarea: Redacción de la memoria.*

| **2.4. Preparación de la defensa** | |
|---|---|
| Descripción | Crear y preparar la presentación, y practicar el discurso para defensa proyecto ante el tribunal, asegurando en lo posible siempre la fluidez y claridad en la comunicación. |
| Duración (h) | Estimada: 64 horas. |
| Recursos necesarios | TBD. |
| Precedente | 2.3. Redacción de la memoria (avanzado), 5. Desarrollo e implementación (finalizado), 6. Pruebas (finalizado). |
| Salidas | Presentación de la defensa finalizada, práctica de la exposición con feedback, confianza en la presentación. |

**Tabla 4.11:** *Descripción de tarea: Preparación de la defensa.*

3. **Recopilación de información**. Se dedica a la investigación y el aprendizaje necesario para poder llevar a cabo el proyecto. Esto incluye la revisión de trabajos, la búsqueda de datos y la formación en tecnologías como Dart/Flutter, smart contracts, AWS, SIGPAC y datos geográficos. Es la fase donde se construye la base de conocimiento técnico y el dominio del proyecto.

| **3.1.1. Lectura de trabajos previos** | |
|---|---|
| Descripción | Revisar artículos científicos, tesis y proyectos existentes relacionados con el tema. |
| Duración (h) | Estimada: 16 horas. |
| Recursos necesarios | Bases de datos académicas como Recolecta [61], Google Scholar y otras tesis de la UPV/EHU mediante ADDI. |
| Precedente | 1.1 (véase 4.2). |
| Salidas | Parte de la bibliografía e información para el análisis. |

**Tabla 4.12:** *Descripción de tarea: Lectura de trabajos previos.*

| **3.1.2. Revisión de antecedentes** | |
|---|---|
| Descripción | Estudiar el contexto agrícola en España y sus problemas, el *state of the art* tecnológico, para justificar la ejecución del proyecto. |
| Duración (h) | Estimada: 16 horas. |
| Recursos necesarios | Acceso a internet y Google Scholar. |
| Precedente | 3.1.1 (véase 4.12). |
| Salidas | Análisis de la situación actual y antecedentes. |

**Tabla 4.13:** *Descripción de tarea: Revisión de antecedentes.*

| **3.1.3. Búsqueda de fuentes de datos** | |
|---|---|
| Descripción | Localizar bases de datos públicas, APIs y similares para usar en el proyecto, asegurando que sea información pública. |
| Duración (h) | Estimada: 16 horas. |
| Recursos necesarios | Acceso a internet y contacto con el MAPA. |
| Precedente | 1.3. (véase 4.4) y 3.1.2. (véase 4.13). |
| Salidas | Lista posibles fuentes de datos, permiso para utilizar toda la información si requiere de alguno. |

**Tabla 4.14:** *Descripción de tarea: Búsqueda de fuentes de datos.*

[61] Enlace a Recolecta: https://buscador.recolecta.fecyt.es/

| 3.2.1. Aprendizaje de Dart y Flutter | |
|---|---|
| Descripción | Adquirir los conocimientos fundamentales y algo más específicos, haciendo apps de prueba y aprendiendo. |
| Duración (h) | Estimada: 32 horas. |
| Recursos necesarios | Videos de Youtube [62], documentación oficial de Flutter y Dart. |
| Precedente | 3.1.3. (véase 4.14). |
| Salidas | Conocimiento sobre la tecnología. |

**Tabla 4.15:** *Descripción de tarea: Aprendizaje de Dart y Flutter.*

| 3.2.2. Aprendizaje de smart contracts | |
|---|---|
| Descripción | Capacitarse en cómo desarrollar un contrato inteligente en Polygon, teniendo en cuenta la seguridad y las vulnerabilidades que pueda haber. |
| Duración (h) | Estimada: 12 horas. |
| Recursos necesarios | Documentación oficial de Solidity [63] y la IDE Remix. |
| Precedente | 1.3 (véase 4.4). |
| Salidas | Conocimiento sobre la tecnología. |

**Tabla 4.16:** *Descripción de tarea: Aprendizaje de smart contracts.*

| 3.2.3. Revisión de conocimientos AWS | |
|---|---|
| Descripción | Repasar y profundizar en los conocimientos de los servicios de AWS que se utilizarán para la infraestructura del proyecto. |
| Duración (h) | Estimada: 8 horas. |
| Recursos necesarios | Documentación oficial de AWS, certificación cloud Certified Cloud Practitioner (CCP), cuenta de AWS. |
| Precedente | Certificación CCP y 1.3. (véase 4.4). |
| Salidas | Repaso sobre la tecnología. |

**Tabla 4.17:** *Descripción de tarea: Revisión de conocimientos AWS.*

| 3.2.4. Aprendizaje de uso de SIGPAC | |
|---|---|
| Descripción | Aprender a utilizar el SIGPAC para la obtención, consulta y análisis de datos geográficos de la península ibérica. |
| Duración (h) | Estimada: 16 horas. |
| Recursos necesarios | Visor SIGPAC, SIGPAC hubcloud y página web del MAPA. |
| Precedente | 1.3. (véase 4.4). |
| Salidas | Conocimiento profundo en el uso de SIGPAC. |

**Tabla 4.18:** *Descripción de tarea: Aprendizaje de uso del sistema SIGPAC.*

[62] Enlaces a algunos tutoriales en Youtube del canal de *Mitch Koko*: https://www.youtube.com/watch?v=zohXXZBUVYI, https://www.youtube.com/watch?v=rYdP2LnBGsA y https://www.youtube.com/watch?v=5XWdLvqi4yY&list=PL1WkZqhlAdC-i3Vs_HBQw9BPT9-_zMSun

[63] Documentación de Solidity, enlace: https://solidity-es.readthedocs.io/es/latest/solidity-by-example.html

| 3.2.5. Aprendizaje de uso de datos geográficos | |
|---|---|
| Descripción | Capacitarse en el manejo de datos espaciales (GIS / OGC) de diversas fuentes, incluyendo la integración con librerías de programación en otros lenguajes. |
| Duración (h) | Estimada: 12 horas. |
| Recursos necesarios | Documentación de FlutterMap y Leaflet, tutoriales de geoprocesamiento cómo: An introduction to the OGC API. Enlace: https://www.youtube.com/watch?v=uswkuZi7g8s |
| Precedente | 1.3. (véase 4.4). |
| Salidas | Competencia en el manejo y análisis de datos geográficos. |

**Tabla 4.19:** *Descripción de tarea: Aprendizaje de uso de datos geográficos.*

4. **Análisis y diseño**. Definición de como funcionará todo el sistema. Se enfoca en entender la lógica de negocio o backend y la estructura general del software o arquitectura. También incluye el diseño de cómo el usuario interactúa con el sistema, haciendo de base para la implementación.

| 4.1. Análisis de la lógica de negocio | |
|---|---|
| Descripción | Comprender cómo funcionarán los procesos internos del sistema, y cómo se intercambiará la información. |
| Duración (h) | Estimada: 8 horas. |
| Recursos necesarios | Visual Studio Code, papel, bolígrafo y PlantUML. |
| Precedente | 1.3. (véase 4.4) y toda la fase 3. |
| Salidas | Diagramas iniciales de bases de datos, de clases / UML. |

**Tabla 4.20:** *Descripción de tarea: Análisis de la lógica de negocio.*

| 4.2. Planteamiento de la arquitectura | |
|---|---|
| Descripción | Diseñar la estructura del sistema, definiendo los componentes principales, sus interacciones, y los patrones de diseño. |
| Duración (h) | Estimada: 16 horas. |
| Recursos necesarios | Conocimiento de patrones y PlantUML. |
| Precedente | 4.1. (véase 4.20), 3.2.3 (véase 4.17). |
| Salidas | Diagrama(s) de arquitectura del sistema. |

**Tabla 4.21:** *Descripción de tarea: Planteamiento de la arquitectura.*

| 4.3. Diseño de la lógica de negocio | |
|---|---|
| Descripción | Traducir la lógica de negocio a un aspecto más técnico, hacer los algoritmos necesarios y la lógica interna. |
| Duración (h) | Estimada: 24 horas. |
| Recursos necesarios | Visual Studio Code, algoritmos de distancia y conocimiento previo. |
| Precedente | 4.1. (véase 4.20). |
| Salidas | Diagramas de clases y bases de datos finales. |

**Tabla 4.22:** *Descripción de tarea: Diseño de la lógica de negocio.*

| **4.4. Diseño de las interfaces** | |
|---|---|
| Descripción | Crear el diseño visual y de interacción de la aplicación. |
| Duración (h) | Estimada: 18 horas. |
| Recursos necesarios | Canva, conocimiento adquirido en 3.2.1. |
| Precedente | 4.1. (véase 4.20). |
| Salidas | Especificaciones de diseño para implementar. |

**Tabla 4.23:** *Descripción de tarea: Diseño de las interfaces.*

5. **Desarrollo e implementación**. Aquí toma forma el trabajo. Consiste en la construcción del software, siguiendo los diseños y análisis previos. Incluye la codificación de las funcionalidades, la implementación del backend, la creación de interfaces, el contrato inteligente, todo lo relacionado con la nube, la conexión a la API de SIGPAC...

| **5.1. Desarrollo de la autenticación** | |
|---|---|
| Descripción | Implementar el sistema de autenticación de usuarios, incluyendo registro, inicio de sesión y recuperación de contraseña por correo electrónico. |
| Duración (h) | Estimada: 32 horas. |
| Recursos necesarios | Firebase Auth, conocimiento adquirido. |
| Precedente | 4.2. (véase 4.21) , 4.3. (véase 4.22) y 3.2.1. (véase 4.15) |
| Salidas | Pantallas de registro, inicio de sesión y método de recuperar contraseña. |

**Tabla 4.24:** *Descripción de tarea: Desarrollo de la autenticación.*

| **5.2. Desarrollo de la lógica de negocio** | |
|---|---|
| Descripción | Codificar las funcionalidades del sistema según el diseño detallado en la lógica de negocio. Interacción con bases de datos, APIs externas y la implementación de algoritmos específicos. |
| Duración (h) | Estimada: 72 horas. |
| Recursos necesarios | Conocimiento previo de Flutter/Dart, bases de datos, patrones y SIGPAC. APIs externas. |
| Precedente | 4.3. (véase 4.22), 3.2.1. (véase 4.15), 3.2.4. (véase 4.18), 3.2.5. (véase 4.19)). |
| Salidas | Lógica de negocio funcional (backend). |

**Tabla 4.25:** *Descripción de tarea: Desarrollo de la lógica de negocio.*

| **5.3. Creación del smart contract** | |
|---|---|
| Descripción | Escribir el código del contrato inteligente utilizando Solidity, prestando especial atención a la seguridad y optimización de gas. |
| Duración (h) | Estimada: 16 horas. |
| Recursos necesarios | Conocimiento de Solidity y una IDE como Remix. |
| Precedente | 3.2.2. (véase 4.16), 4.3. (véase 4.22). |
| Salidas | Código fuente del contrato, verificado por Polygon. |

**Tabla 4.26:** *Descripción de tarea: Creación del smart contract.*

| **5.4. Despliegue del smart contract** | |
|---|---|
| Descripción | Poner en funcionamiento el contrato inteligente. |
| Duración (h) | Estimada: 12 horas. |
| Recursos necesarios | Metamask, Remix, una cuenta con criptomoneda suficiente para el despliegue y las comisiones. |
| Precedente | 5.3. (véase 4.26), 3.2.2. (véase 4.16). |
| Salidas | Smart contract desplegado con eventos probados y verificado en la red. |

**Tabla 4.27:** *Descripción de tarea: Despliegue del smart contract.*

| **5.5. Desarrollo de las interfaces** | |
|---|---|
| Descripción | Codificar la interfaz de usuario de la aplicación. |
| Duración (h) | Estimada: 36 horas. |
| Recursos necesarios | Emulador, teléfono móvil, Visual Studio Code y conocimiento adquirido. |
| Precedente | 4.4. (véase 4.23), 3.2.1. (véase 4.15). |
| Salidas | Interfaz de usuario implementada y funcional, componentes visuales conectados con el backend. |

**Tabla 4.28:** *Descripción de tarea: Desarrollo de las interfaces.*

| **5.6. Implementación de la arquitectura final en AWS** | |
|---|---|
| Descripción | Configurar y desplegar la infraestructura, S3, DynamoDB, Lambdas, API Gateway, IAM, SSM y otros servicios necesarios en AWS. |
| Duración (h) | Estimada: 44 horas. |
| Recursos necesarios | Cuenta AWS, conocimiento adquirido y código fuente. |
| Precedente | 4.2.(véase 4.21), 3.2.3. (véase 4.17). |
| Salidas | Infraestructura de AWS configurada y operativa, información accesible desde la app. |

**Tabla 4.29:** *Descripción de tarea: Implementación de la arquitectura final en AWS.*

6. **Pruebas**. Son la clave para que la aplicación funcione como debe y que cualquier posible fallo se resuelva antes de llegar al público. Para ello, se plantean 3 tipos de pruebas separadas en dos tareas distintas, donde se desarrollarán unas pruebas unitarias, otras automatizadas y otras de usuario.

| **6.1. Pruebas técnicas** | |
|---|---|
| Descripción | Plantear y codificar todas las pruebas técnicas, incluyendo también algunos de los tests en múltiples dispositivos. En este apartado se encuentran tanto los *tests* unitarios como los automatizados. |
| Duración (h) | Estimada: 24 horas. |
| Recursos necesarios | Visual Studio Code, Robo Test, dispositivo móvil y emulador. |
| Precedente | 5.6. (véase 4.29). |
| Salidas | Ficheros de pruebas e informes de Robo Test. |

**Tabla 4.30:** *Descripción de tarea: Pruebas técnicas.*

| **6.2. Pruebas de usuario** | |
|---|---|
| Descripción | Pensar y ejecutar las pruebas de usuario, teniendo en cuenta cuál es el objetivo o *target* de la aplicación. |
| Duración (h) | Estimada: 24 horas. |
| Recursos necesarios | Contactar con agricultores y voluntarios, varios dispositivos móviles, papel y bolígrafo. |
| Precedente | 6.1. (véase 4.31). |
| Salidas | Feedback y opiniones de usuarios. |

**Tabla 4.31:** *Descripción de tarea: Pruebas de usuario.*

## 4.3. Planificación temporal

Para esta sección se ha estimado una carga de trabajo flexible en el que los días trabajados son los **laborables (L-V) y el sábado**, en la que dependiendo de la época hay unas horas u otras. Esto se define a continuación en la figura 4.2, y es en base a la disponibilidad del autor para dedicar al Trabajo de Fin de Grado, incluyendo las prácticas extracurriculares, los días de clase y el tiempo a dedicación completa.


**Figura 4.2:** *Carga de trabajo en base a las fechas. Hecho en GanttPRO.*

Una vez conocidas las tareas a realizar, hay que decidir cómo se van a distribuir en el tiempo y estimar la duración de cada una, que se representa mediante un **diagrama de Gantt** [64] semanal, separado en dos figuras (4.3 y 4.4) para mayor claridad. Para poder ver la continuidad, se incluye una figura (4.5) con otro Gantt mensual. Además, se ha creado otro diagrama semanal pero completo, que está disponible en: https://drive.google.com/file/d/1IrRrjQsCKzu7a0g8lrvDMemN-o7X7eOG/view?usp=sharing debido a que en la documentación no se vería al completo.

Los diagramas se han dividido de acuerdo a las tareas generadas en el EDT, y a la izquierda del diagrama se encuentra el listado con todas las tareas. La elección del tema es la primera tarea a desarrollar y por tanto, cuando se termine esa se pasará a la siguiente, de esta forma, se sigue un proceso en serie tarea tras tarea que inicia desde el comienzo del proyecto hasta el final. Nótese que tanto las reuniones de seguimiento como la redacción son tareas de **elaboración contínua**, mejor conocidas como tareas *hammock* o hamaca.

[64] Los diagramas de Gantt sirven para visualizar los componentes básicos del proyecto y poder organizar cuáles se pueden completar en que instante.

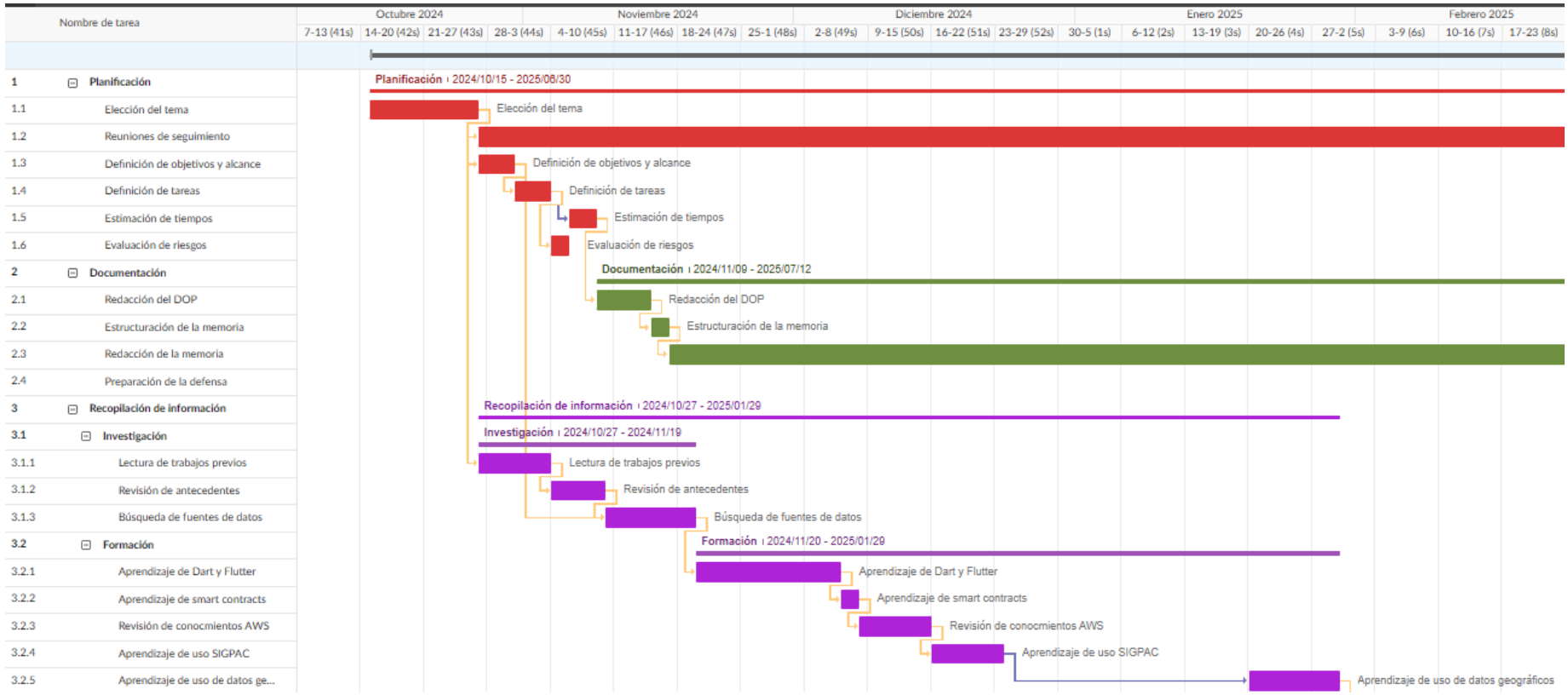



**Figura 4.3:** *Diagrama Gantt por semanas, parte 1. Hecho con GanttPRO.*

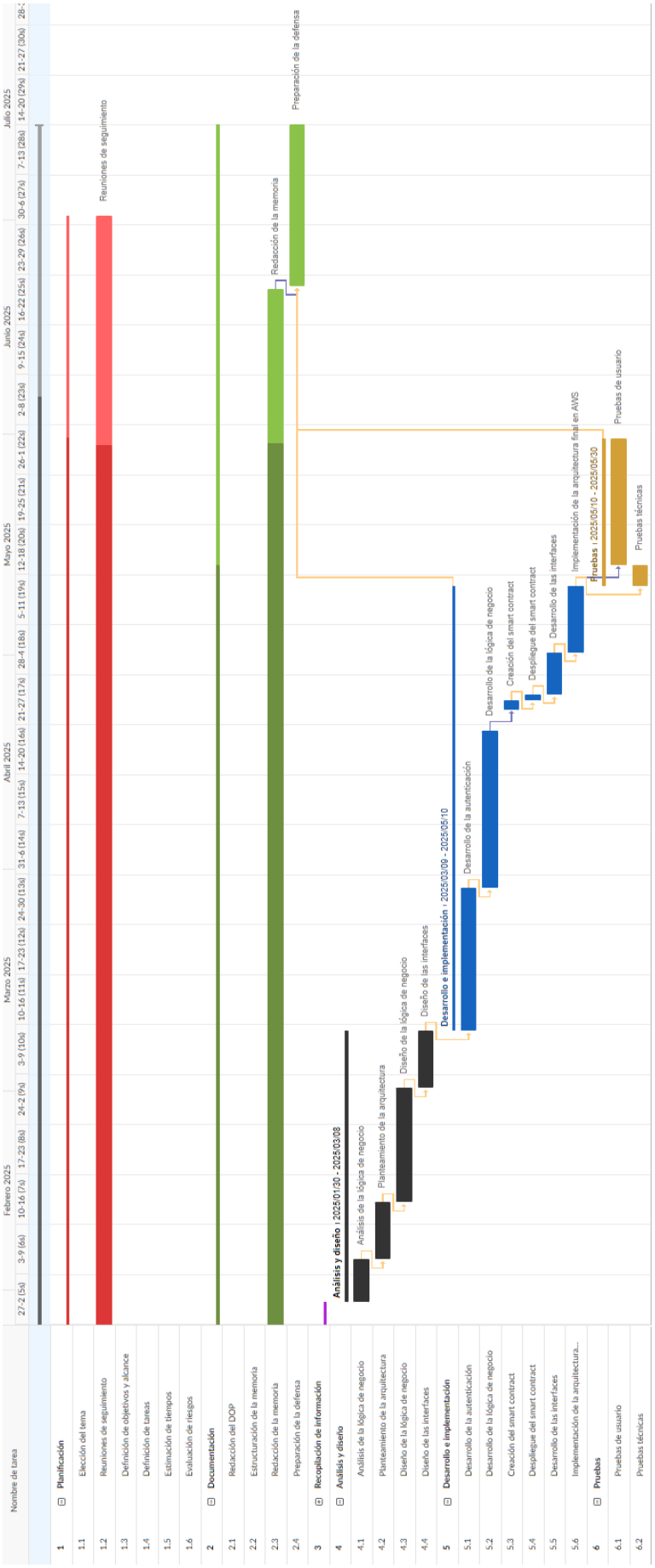



**Figura 4.4:** *Diagrama Gantt por semanas, parte 2. Hecho con GanttPRO.*

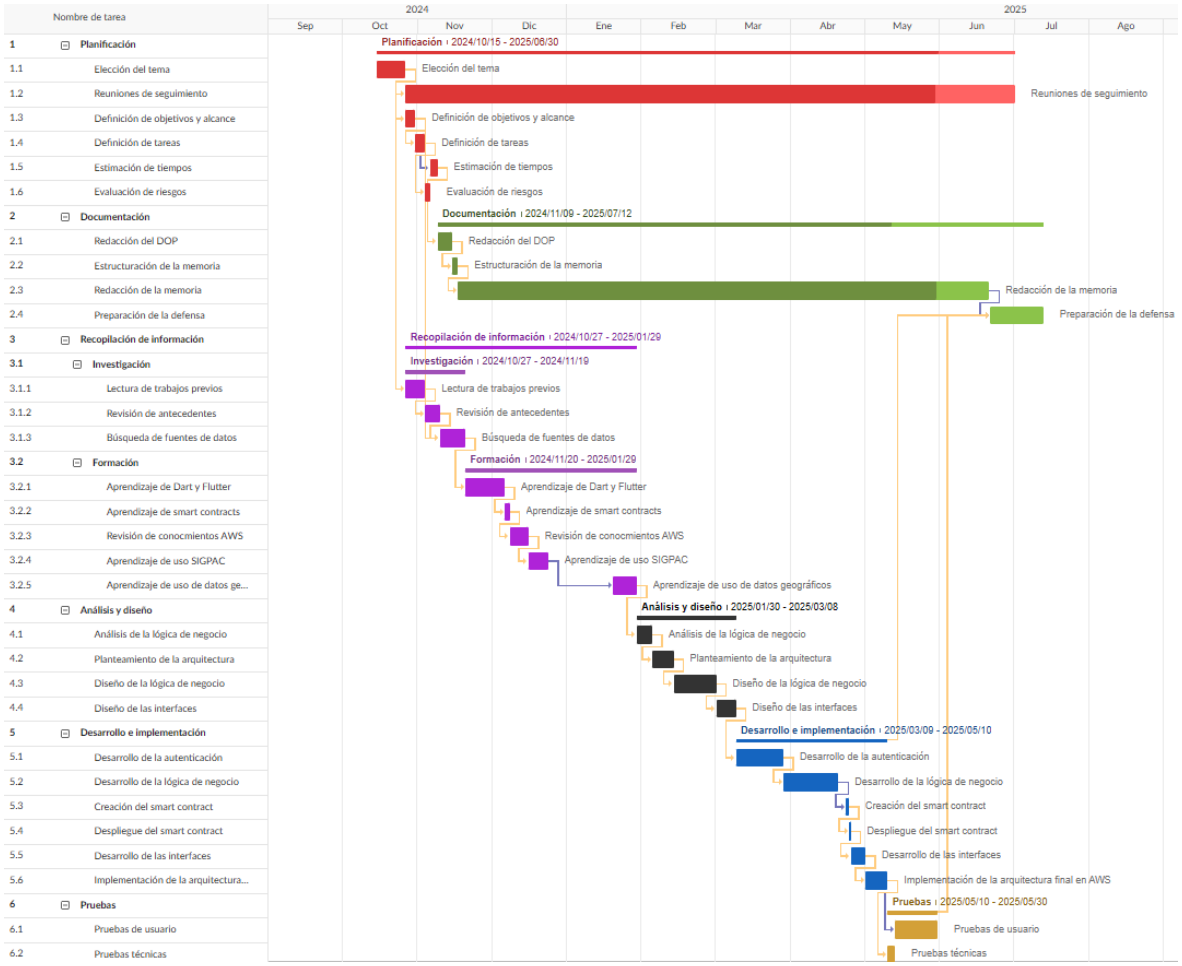



**Figura 4.5:** *Diagrama Gantt por meses Hecho con GanttPRO.*

Además, para el cálculo de la evaluación económica del proyecto, es conveniente tener una tabla cómo la 4.32, con las horas estimadas de cada tarea, para calcular el gasto de personal hipotético del proyecto.

| **Sección** | **Duración estimada** |
|---|---|
| Planificación | 46 h |
| Documentación | 164 h |
| Recopilación de información | 128 h |
| Análisis y diseño | 66 h |
| Desarrollo e implementación | 212 h |
| Pruebas | 48 h |
| **Total** | **664 h** |

**Tabla 4.32:** *Horas estimadas por cada sección del EDT.*

## 4.4. Evaluación económica

En esta sección se hará un análisis de costes y posibles beneficios económicos que conlleva AgroFind, agrupados por tipo de gasto correspondiente.

### Costes de personal

Estos representan el gasto relacionado con los recursos humanos involucrados en el proyecto, que en la mayoría de proyectos lo más costoso. Para calcular el coste de personal, se tiene en cuenta el sueldo promedio de un ingeniero informático junior en España, unos **12,67 €/h** [77]. Por otra parte, el salario medio en la UPV/EHU para un docente de universidad es de **21,65€/h** [78]. Las horas de reuniones se computan para ambos individuos, y se añaden otras 16 horas a las 12 asignadas en las tareas, que en total son 28 horas, al docente de trabajo de lectura, feedback y gestión del proyecto.

Con esos datos se puede calcular el coste de personal: $\sum_{i=1}^{n}(Horas_i \cdot Sueldo/hora_i)$

$$\textbf{Coste de personal} = 664\text{h} \cdot 12{,}67€/\text{h} + 28\text{h} \cdot 21{,}65€/\text{h} = \textbf{9017.68}€$$

### Costes de hardware

En esta sección se incluye la inversión económica inicial para disponer de los equipos y el coste para mantenerlos, los dispositivos (sin tener en cuenta teclado, ratón y auriculares, puesto que son una fracción no relevante del coste) que se han usado han sido los siguientes, donde el precio es el de adquisición en el momento que se compró cada uno de ellos (tabla 4.33):

| Descripción | Precio | Amortización | Coste en 9 meses |
|---|---|---|---|
| Ordenador de sobremesa | 978 € | 6 años [65] | 122.25 € |
| Teléfono móvil Xiaomi Mi A2 | 270 € | 3 años | 67.50 € |
| Pantalla Phillips 223V7 | 125 € | 4 años | 23.44 € |
| **Total:** | - | - | **213.19 €** |

**Tabla 4.33:** *Coste total del hardware utilizado en el proyecto.*

[65] Un ordenador de sobremesa suele amortizar a 5 años, pero este dispositivo lleva funcionando 6 años sin ningún problema. Las especificaciones del mismo:
AMD Ryzen 5 3600X, 16GB RAM, 500GB SSD Kingston SA2000M, GeForce GTX 1660 6 GB.

## Costes de software

Respecto al software utilizado, hay varios aspectos que pueden variar respecto a su uso gratuito, que en caso de hacer la app funcional en el mercado con un cierto número de usuarios, habría que pagarlos (tabla 4.34). Datos obtenidos de Firebase Pricing [79], WeatherAPI Pricing [80] y AWS Calculator [81]. La aplicación puede funcionar de manera gratuita hasta 300 usuarios. Esto se calcula utilizando AWS Calculator, donde se pueden crear estimaciones personalizadas en base a los servicios utilizados, en los que para 300 usuarios (proyecto piloto) se ha determinado que AgroFind funcionaría en la capa gratuita y para el resto, se han creado dos estimados personalizados: 2000 usuarios [82] (fase inicial) y 30000 usuarios [83] (fase de expansión).

| Servicio | **300 usuarios** | **2k usuarios** | **30k usuarios** | Notas |
|---|---|---|---|---|
| Firebase | 0 € | 0 € | 0 € | 0.0055€/mes por usuario a partir de 50.000. |
| AWS | 0 € | 5,15 € | 93,50 € | Según el estimado. |
| WeatherAPI | 0 € | 0 € | 25 € | Con el plan Pro. |
| **Total:** | **0 €** | **5,15 €/mes** | **118,50 €/mes** | |

**Tabla 4.34:** *Comparativa de costes estimados del software para distintos volúmenes de usuarios activos.*

## Coste total estimado

A los gastos previamente obtenidos, hay que añadir los llamados **gastos indirectos** o costes de mantenimiento de la elaboración del proyecto (electricidad, agua, internet...) y se ha considerado que suponen un 3 % del coste total de proyecto. El coste final total del proyecto se desglosa y calcula cómo en la tabla 4.35, donde el coste de software será de 0€, debido a que no ha supuesto ninguno porque AgroFind no se ha puesto en producción.

| **Concepto** | **Importe (€)** |
|---|---|
| Costes directos - Personal | 9.017,68 € |
| Costes directos - Hardware | 213,19 € |
| **Subtotal costes directos** ($SCD$) | **9.230,87 €** |
| Gastos indirectos (3 %, $GI$) | $SCD * 0{,}03 = 276,92$ |
| **Coste total estimado** (CTE) | $GI + SCI =$ **9.507,79 €** |

**Tabla 4.35:** *Resumen del coste estimado del proyecto.*

## Posibles beneficios

Al ser una prueba de concepto, es complicado calcular cuáles serían los beneficios posibles que la aplicación pudiera reportar, aunque en caso de no haber interés por ninguna de las opciones que se presentan a continuación, el autor expresa su voluntad de hacer el proyecto público con licencia GNU General Public License v3.0 (GPL) [66].

- **Venta o Licencia de la Propiedad Intelectual:**
  - *Adquisición por un Organismo Público:* Venta de la idea, del prototipo actual (MVP) o licencia de uso a entidades como el MAPA o el Gobierno Vasco.
  - *Contrato de Desarrollo:* Posibilidad de ser contratado para desarrollar y mantener la aplicación a gran escala en un organismo público o privado.
- **Creación de una Startup:** Lanzar el producto al mercado como una SaaS (véase sección 3.2.2) y buscar inversión para escalar el negocio y poderlo ejecutar. Para ello habría que hacer algunos cambios al proyecto, pero podría ser una opción lucrativa a largo plazo.
- **Mejora del Perfil Profesional:**
  - *Portafolio y CV:* Un proyecto que demuestra las habilidades y capacidad de innovación del autor, mejorando el atractivo en el mercado laboral.
  - *Networking y Especialización:* Creación de contactos clave en sectores como el agro-tecnológico, cloud o blockchain.
- **Becas, Premios o Concursos:** Oportunidad de obtener fondos a través de concursos de innovación o programas de financiación pública/europea. La propia universidad ofrece becas a través de empresas interesadas en trabajos de fin de grado, por tanto, esto podría ser una opción para desarrollar el proyecto más todavía.

[66] Esta permite el uso, estudio, modificación y distribución del software, siempre que se mantenga bajo la misma licencia GPL. No permite la distribución de programas sin el código fuente o sin una oferta de cómo obtenerlo. Enlace a la licencia: https://www.gnu.org/licenses/gpl-3.0.html

## 4.5. Gestión de riesgos

Para evitar pérdidas potenciales durante el transcurso del proyecto, es importante generar un **plan de contingencia** para poder mitigar al máximo los efectos que estos puedan tener, así como prevenir las causas que puedan llevar a que sucedan. Por la subjetividad que supone calcular de la probabilidad de determinados riesgos, se ha diseñado una tabla orientativa (4.36) para tener como modelo a seguir.

| Probabilidad | Descripción |
|---|---|
| **Muy Baja** (0-10 %) | Es muy poco probable que ocurra. |
| **Baja** (10-30 %) | Podría ocurrir, pero no es frecuente. |
| **Media** (30-60 %) | Eventos que se dan ocasionalmente. |
| **Alta** (60-100 %) | Es probable que ocurra. |

**Tabla 4.36:** *Clasificación de la probabilidad de riesgos.*

Otro factor a tener en cuenta es el impacto. Habrá riesgos que no supongan demasiado, pero otros, si se produjeran, podrían retrasar el proyecto de forma irreversible. Al igual que sucede con las probabilidades, se ha generado otra tabla (4.37) con el tipo de impacto y el retraso asociado al mismo.

| Impacto | Descripción | Retraso Estimado |
|---|---|---|
| **Bajo** | Pequeñas desviaciones. | 1 - 3 días |
| **Medio** | Desviaciones notables. | 3 - 7 días |
| **Alto** | Impacto significativo, comprometiendo tareas. | 1 - 4 semanas |
| **Crítico** | El proyecto se paraliza o reestructura. | $>$ 4 semanas |

**Tabla 4.37:** *Clasificación del impacto de riesgos y su retraso estimado.*

En base a estos criterios, probabilidad e impacto, hay que generar tanto el plan de contingencia pertinente como el plan de prevención correspondiente para cada posible riesgo, donde para cada uno se detalla a continuación.

| **Cambio en el SDK de Flutter** | |
|---|---|
| **Prevención** | Estar al corriente de las funcionalidades a las que se dejará de dar soporte. |
| **Plan de contingencia** | No actualizar de versión o cambiar las funcionalidades. |
| **Probabilidad** | Muy baja |
| **Impacto** | Medio |

**Tabla 4.38:** *Prevención y contingencia: Cambio en el SDK de Flutter.*

| **Enfermedad o incapacidad prolongada** | |
|---|---|
| **Prevención** | Mantener un estilo de vida saludable y asegurar períodos de descanso adecuados durante todo el proyecto. |
| **Plan de contingencia** | Establecer un contacto de emergencia que pueda acceder a la documentación y repositorios en caso necesario. |
| **Probabilidad** | Muy baja |
| **Impacto** | Crítico |

**Tabla 4.39:** *Prevención y contingencia: Enfermedad o incapacidad prolongada.*

| **Enfermedad o incapacidad breve** | |
|---|---|
| **Prevención** | Tomar precauciones ante la gripe y cuidar las muñecas y brazos ante posibles molestias. |
| **Plan de contingencia** | Acudir al médico en caso necesario, acompañado de reposo tanto físico como mental para evitar *burnout*. |
| **Probabilidad** | Alta |
| **Impacto** | Bajo |

**Tabla 4.40:** *Prevención y contingencia: Enfermedad o incapacidad breve.*

| **Fallo de ordenador de sobremesa o periféricos** | |
|---|---|
| **Prevención** | Utilizar el dispositivo a una temperatura adecuada, haciendo análisis de estado y desgaste. Cuidar todos los periféricos. |
| **Plan de contingencia** | Disponer de un equipo de respaldo, en este caso un portátil, y algún periférico alternativo. |
| **Probabilidad** | Baja |
| **Impacto** | Medio |

**Tabla 4.41:** *Prevención y contingencia: Fallo de ordenador de sobremesa o periféricos.*

| **Fallo o caída de herramientas auxiliares necesarias** | |
|---|---|
| **Prevención** | Mantener el código y la documentación en un sistema de control de versiones con repositorio remoto como GitHub. Tener herramientas alternativas en caso de fallo prolongado. |
| **Plan de contingencia** | Tener copias de seguridad de los datos de trabajo y configuraciones importantes en la nube, y un plan para reinstalar el entorno de desarrollo rápidamente. |
| **Probabilidad** | Media |
| **Impacto** | Medio |

**Tabla 4.42:** *Prevención y contingencia: Fallo o caída de herramientas auxiliares necesarias.*

| **Falta de conocimientos técnicos bloqueantes** | |
|---|---|
| **Prevención** | Realizar una formación previa en las tecnologías clave y no subestimar la complejidad de las funcionalidades. |
| **Plan de contingencia** | Dedicar tiempo específico al aprendizaje, buscar soluciones en *Stack Overflow*, preguntar a la IA o simplificar la funcionalidad. |
| **Probabilidad** | Alta |
| **Impacto** | Bajo |

**Tabla 4.43:** *Prevención y contingencia: Falta de conocimientos técnicos bloqueantes.*

| **Planificación temporal errónea o subestimación de tarea** | |
|---|---|
| **Prevención** | Desglosar el proyecto en tareas pequeñas y revisar periódicamente el progreso frente al plan original. |
| **Plan de contingencia** | Volver a priorizar las funcionalidades para asegurar tener un MVP, comunicar cualquier desviación de plazo de forma temprana al tutor. |
| **Probabilidad** | Alta |
| **Impacto** | Variable, en base al caso. |

**Tabla 4.44:** *Prevención y contingencia: Planificación temporal errónea o subestimación de tarea.*

| **Cambio del alcance del proyecto** | |
|---|---|
| **Prevención** | Definir de forma muy estricta el objetivo y alcance, dejar para el final las funcionalidades no esenciales. |
| **Plan de contingencia** | Documentar cualquier nueva idea o requisito, evaluar su impacto en el tiempo y el esfuerzo, y posponer su implementación. |
| **Probabilidad** | Alta |
| **Impacto** | Medio |

**Tabla 4.45:** *Prevención y contingencia: Cambio del alcance del proyecto.*

| **Caída de servicios cloud como Firebase o AWS** | |
|---|---|
| **Prevención** | Seleccionar proveedores de servicios cloud con alta fiabilidad, diseñar la aplicación para manejar fallos temporales. |
| **Plan de contingencia** | Considerar una migración a un servicio alternativo si la interrupción es prolongada. |
| **Probabilidad** | Baja |
| **Impacto** | Alto |

**Tabla 4.46:** *Prevención y contingencia: Caída de servicios cloud como Firebase o AWS.*

| **Caída de alguna API (SIGPAC, WeatherAPI...)** | |
|---|---|
| **Prevención** | Estar al tanto de todas las posibles actualizaciones en estos servicios y utilizarlos de manera responsable para no evitar sobrecargas en los mismos. |
| **Plan de contingencia** | Plantear una migración a un servicio similar (si es posible) en caso de que la interrupción se alargue en el tiempo. |
| **Probabilidad** | Media |
| **Impacto** | Medio |

**Tabla 4.47:** *Prevención y contingencia: Caída de alguna API (SIGPAC, WeatherAPI...).*

| **Apagón eléctrico [67] o interrupción de conexión a internet** | |
|---|---|
| **Prevención** | Guardar el trabajo con mucha frecuencia, tener un dispositivo siempre con batería, y conexión a internet alternativa si la disponibilidad es crítica. |
| **Plan de contingencia** | Planificar trabajar en una ubicación alternativa (biblioteca o similares) si la interrupción es local, en caso de ser de forma masiva, notificarlo lo antes posible al tutor del trabajo. |
| **Probabilidad** | Baja |
| **Impacto** | Medio |

**Tabla 4.48:** *Prevención y contingencia: Apagón eléctrico o interrupción de conexión a internet.*

| **Pérdida irreversible de código o documentación** | |
|---|---|
| **Prevención** | Uso habitual de herramientas de control de versiones como Github, descargar lo editado cada día de Overleaf y copias de seguridad regulares almacenadas en la nube y en distintos dispositivos. |
| **Plan de contingencia** | Restaurar la documentación desde la última copia de seguridad o el código desde el repositorio. Si hay pérdida parcial, priorizar la recreación de la parte más crítica. |
| **Probabilidad** | Muy Baja |
| **Impacto** | Crítico |

**Tabla 4.49:** *Prevención y contingencia: Pérdida irreversible de código o documentación.*

[67] Se registró un apagón eléctrico masivo en la Península Ibérica el 28 de abril de 2025. Esto afectó a la región de España de AWS (eu-south-2) durante unos instantes, pero se recuperó rápidamente debido a los sistemas de alimentación autónomos, mientras que al autor le afectó durante 4 horas. Por otra parte, SIGPAC dejó de funcionar durante 2 días aproximadamente, lo que causó un retraso notable en el proyecto. Fuente: mediciones propias.

# 5. Análisis y diseño

En este capítulo de analizará todo lo necesario para poder realizar la implementación final, desde la captura de requisitos, la redacción de los casos de uso y del modelo de dominio. Finalmente, se explicará en detalle la arquitectura de todo el sistema, desde el frontend hasta el backend incluyendo las sendas bases de datos en ambas secciones. Hay que mencionar que al hacer reuniones de seguimiento con objetivos locales, indirectamente se ha empleado **Scrum** [68] como marco de trabajo para el proyecto.

## 5.1. Captura de requisitos

En este proceso del análisis se han identificado las necesidades y expectativas de los usuarios y por tanto, se han distinguido dos tipos de requisitos:

### 5.1.1. Requisitos funcionales

Estos son los que describen las funciones concretas que el sistema debe cumplir, el comportamiento de la aplicación, cómo responde la misma ante los *inputs* de los usuarios, y en que condiciones ocurre esto:

- **Inicio de la aplicación:**
  - AgroFind debe de comprobar si es la primera vez que el usuario abre la aplicación. En ese caso, debe de mostrar la pantalla de registro.
  - En caso contrario, debe comprobar si el usuario ha iniciado sesión anteriormente y hay una *cookie de sesión* válida. Si se cumple esto, se inicia sesión directamente. Si no, se debe de mostrar la pantalla de *login*.
- **Registro:**
  - El usuario se debe de poder registrar con un nombre y apellidos, correo electrónico y una contraseña.
  - Por seguridad, el sistema debe de comprobar que la contraseña sea lo suficientemente segura, que el usuario no esté registrado previamente y que no haya ningún campo vacío. En cualquiera de estos tres casos, se le hará saber esto al agricultor por pantalla.
- **Inicio de sesión o *login*:**
  - El agricultor debe de poder iniciar sesión usando el correo electrónico y la contraseña.

[68] Scrum es marco de trabajo ágil que ayuda a los equipos a gestionar proyectos de forma eficiente con conceptos como: *sprints*, *artefactos*, *reuniones frecuentes*... enlace a la web oficial: scrum.org

  - En caso de olvidar la contraseña, el usuario puede solicitar la recuperación de la misma a través de su correo electrónico. Le deberá de llegar un correo personalizado con una contraseña temporal.

- **Gestión de eventos:**
  - Si el agricultor posee alguna parcela reclamada, entonces deberá de poder añadir nuevos eventos, definir su tipo e información relevante sobre los mismos.
  - El sistema deberá comprobar que la parcela en la que se quiere añadir el evento es cultivable y que el usuario está a menos de 2km de la misma.
  - Cada usuario no tendrá datos sobre los eventos de otros usuarios.
  - El agricultor tiene que poder ver según abre la aplicación cuáles han sido sus eventos ordenados del más reciente al más antiguo. Además, debe de poder ver en que parcela ha sido y clicar en el nombre de la parcela para añadir nuevos eventos sin tener que entrar al mapa.
- **Gestión de parcelas:**
  - Se debe de poder añadir parcelas nuevas a la lista de parcelas del usuario, y éste tiene que tener la capacidad de ver los detalles de las mismas.
  - AgroFind debe comprobar que la parcela que un usuario intenta añadir no es de nadie, y además, no debe almacenar información detallada sobre las parcelas de otros usuarios.
  - El usuario podrá cambiar el nombre de sus parcelas, pero no podrá eliminar una parcela existente, debido a que si la ha podido añadir, ningún otro agricultor la debería de añadir.
- **Mapas:**
  - Debe de haber dos mapas, uno con las parcelas del usuario y otro con las parcelas reclamadas por todos y los recintos SIGPAC.
  - Una vez cargado uno de los mapas, si se va a otra pestaña y se vuelve, el mapa estará listo instantáneamente. El único *delay* que puede suceder es al cargar los recintos SIGPAC, por la conexión a internet.
- **Tiempo:**
  - El usuario debe de poder consultar el tiempo actual si tiene conexión a internet.
  - AgroFind debe de gestionar el tiempo en cada parcela, y dependiendo de la distancia a la que esté cada parcela, poner la misma temperatura sin enviar más peticiones.

- **Cambio de tema:**
  - Se debe de poder cambiar de tema claro a tema oscuro.
  - El cambio de tema debe de permanecer aún habiendo cerrado y abierto la aplicación, y debe de afectar a la totalidad de la misma, antes y después del inicio de sesión.
- **Generación de códigos QR:**
  - El labrador debe de poder generar un QR al cosechar su parcela, que certifique mediante la blockchain que esa cosecha se ha realizado en esa ubicación, en esa fecha, y que anteriormente había plantado o sembrado lo mismo que se está cosechando.
  - El informe al que se accede mediante el código QR tiene que estar accesible públicamente para que aún sin tener el código, con el enlace se pueda acceder al mismo.
- **Sin servidor:**
  - Debe de ser una aplicación serverless al 100 %. Esto implica que no debe de depender de un único servidor, sino de varios servicios que realizan ciertas funciones.
  - Todo lo que se realice en el backend tiene que tener una latencia aceptable, es decir, el tiempo de respuesta ante obtener las parcelas, los eventos o añadir un nuevo evento a la blockchain de un servicio debe de ser lo más rápido posible.
  - Específicamente, la API propia debe de funcionar de manera segura, privada y sin sobresaltos, utilizando las herramientas que sean necesarias para ello.
- **Trazabilidad completa:**
  - La aplicación debe de tener todos los datos necesarios para realizar la trazabilidad completa de los eventos y las parcelas, tanto para poder generar los informes al cosechar como para visualizar los eventos.
  - Es necesario emplear tanto la blockchain como una tabla en alguna base de datos que almacene toda esta información, para no tener que consultar a la blockchain por cada evento, porque esto ralentizaría la aplicación.

### 5.1.2. Requisitos no funcionales

Los requisitos no funcionales son aquellos que explican las características del sistema que no están directamente relacionadas con las funciones, sino con la forma en la que estas se llevan a cabo.

- **Usabilidad:**
  - AgroFind debe de ser accesible para los agricultores, con un diseño uniforme a lo largo de la aplicación, teniendo en cuenta siempre la paleta de colores específica.
  - La interfaz de usuario de la aplicación debe de ser lo más intuitiva y fácil de usar posible, para que los labradores puedan usarla sin restricciones de conocimiento tecnológico. La autenticación tiene que dar opciones a recuperar la contraseña en caso de olvido
- **Compatibilidad:**
  - Debe de ser compatible en su totalidad con la gran mayoría de los dispositivos Android, y tiene que poderse adaptar en el futuro a iOS, así como con los navegadores modernos en su versión web y Windows, Linux y macOS en su versión de escritorio.
  - No debe de haber impedimentos para usar AgroFind desde cualquier dispositivo, y no puede haber funcionalidades inexistentes o erróneas. Puede haber cambio en las interfaces entre un sistema y otro, pero nada que afecte a la funcionalidad.
- **Seguridad.**
  - *Confidencialidad*: Los agricultores no tienen que compartir sus datos con ningún otro tercero, y se protege el acceso a las bases de datos en la nube de accesos no deseados a la información. Toda la información que se almacena en la blockchain debe de estar encriptada u ofuscada de alguna manera.
  - *Autenticación*: Solo los usuarios que han iniciado sesión pueden acceder a la aplicación.
  - *Integridad*: Hay que asegurar que los datos no puedan ser modificados sin autorización.
- **Rendimiento:**
  - La aplicación debe de ser agradable de usar, y por tanto no debe haber esperas a no ser que sean estrictamente necesarias (como pasará al cosechar y generar el PDF, figura 6.43).
  - AgroFind no debe de consumir demasiados recursos, está pensado para funcionar en dispositivos relativamente antiguos (hasta 6 años de antigüedad aproximadamente). Se debe de probar en varios dispositivos diferentes.
  - No solo la aplicación, sino todos los servicios gestionados por AgroFind o de terceros tienen que tener una velocidad aceptable para hacer la app lo más atractiva para los agricultores.

### 5.1.3. Casos de uso

Los **diagramas de casos de uso** sirven para especificar la comunicación y comportamiento de un sistema mediante la interacción con sus usuarios. Además, la **jerarquía de actores** de los casos de uso se relaciona con la categorización de los usuarios que interactúan con el sistema en distintos niveles. A continuación se incluyen dos diagramas, uno con la jerarquía de actores en la figura 5.1 y otro con los casos de uso al completo en la figura 5.2.


**Figura 5.1:** *Jerarquía de actores de AgroFind.*

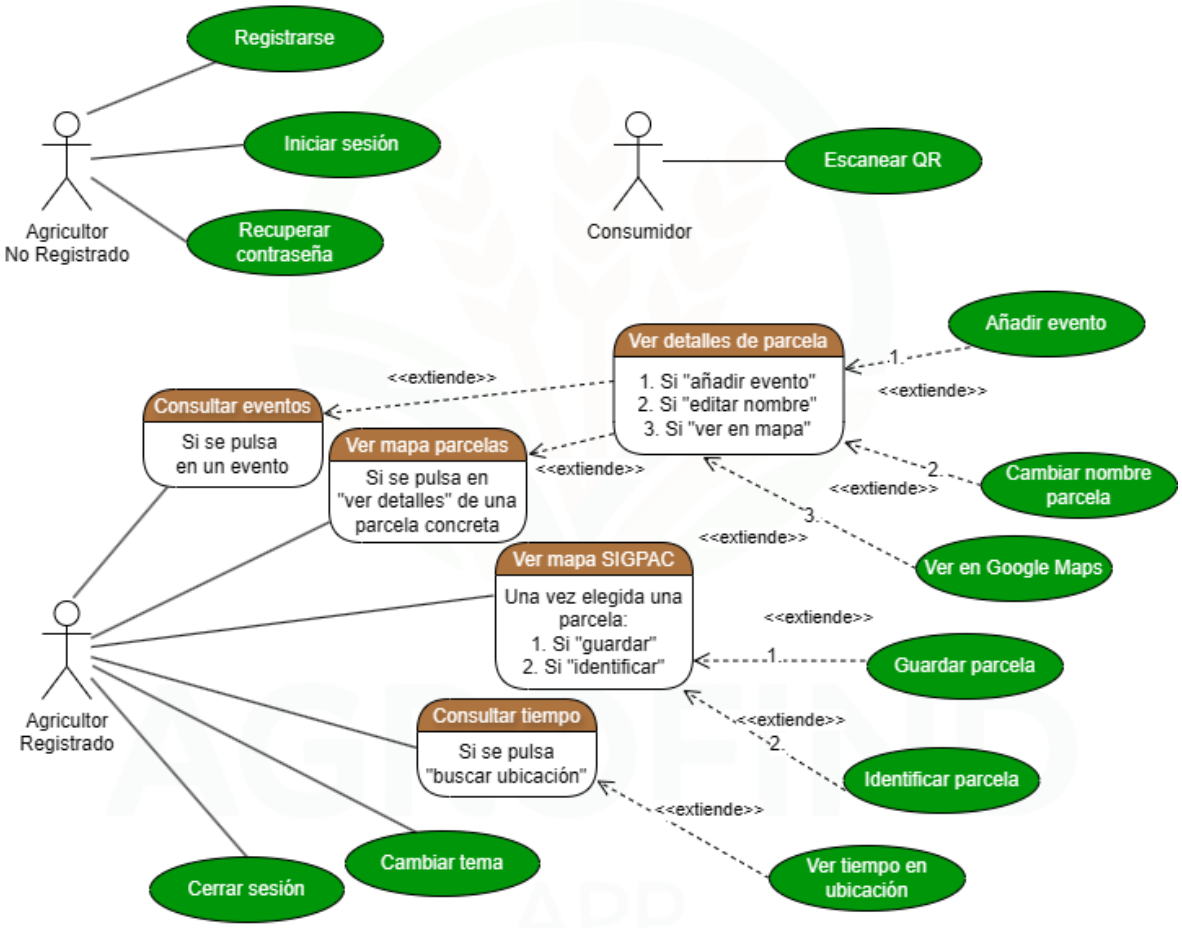



**Figura 5.2:** *Diagrama de casos de uso AgroFind.*

**Actor: Consumidor**

- **Escanear QR:** El consumidor final utiliza la cámara de su dispositivo (sin ninguna aplicación necesaria) para escanear códigos QR presentes en los productos, con el fin de obtener la información relacionada a la trazabilidad.

**Actor: Agricultor No Registrado**

- **Registrarse:** Permite a un nuevo usuario crear una cuenta en el sistema de Agrofind proporcionando la información necesaria (nombre y apellidos, correo electrónico y contraseña). Este es el primer paso para acceder a las funcionalidades de un agricultor registrado.
- **Iniciar sesión:** Acceso al sistema para usuarios ya registrados mediante la validación de credenciales. Cuando un agricultor no registrado inicia sesión pasa a ser agricultor registrado.
- **Recuperar contraseña:** Ofrece un mecanismo para que el agricultor que ha olvidado su contraseña pueda restablecer su acceso a la cuenta, a través de un proceso de verificación por correo electrónico (este proceso lo maneja Firebase).

**Actor: Agricultor Registrado**

El agricultor en esta sección se asume que está registrado y que ha iniciado sesión.

- **Consultar eventos:**
  - Permite al agricultor registrado revisar la lista de eventos importantes (siembras, arados, riegos, fumigaciones, etc.) asociados a sus parcelas.
  - Extiende a *Ver detalles de parcela.*
  - **Ver mapa parcelas:**
    - Permite al agricultor registrado visualizar un mapa interactivo con todas sus parcelas registradas, mostrando su silueta mediante polígonos por encima de una *ortofoto* (véase 2.2) o imagen del terreno. Este caso de uso es punto de entrada directo para poder ver los detalles de cada una.
    - Extiende a *Ver detalles de parcela.*
- **Ver detalles de parcela:**
  - El agricultor registrado accede a información detallada sobre una parcela específica, incluyendo datos relevantes como el nombre, el identificador único, las coordenadas, la extensión en hectáreas, su tipo de uso según SIGPAC...
  - *Extensiones:*
    - **Añadir evento:** El agricultor puede registrar un evento relacionado con la parcela, siempre y cuando cumpla con las condiciones de estar a menos de 2 km de la misma y que la parcela sea susceptible de ser usada

para fines agrícolas (si el uso SIGPAC cumple con las condiciones [69]). El evento contiene el identificador de la parcela, el tipo de evento (siembra, cosecha, arado...) e información específica del evento. Cosecha tendría la semilla plantada previamente, el arado tendría el tipo de arado y así con todos los tipos.

- **Cambiar nombre parcela:** Se puede cambiar el nombre de la parcela para poder organizarlas alfabéticamente o simplemente por dar un nombre distintivo a cada una.

- **Ver en Google Maps:** Permite visualizar la ubicación geográfica de la parcela directamente en la aplicación de Google Maps para poder navegar hacia ella usando GPS.

- **Ver mapa SIGPAC:**

  - Proporciona acceso a la visualización de parcelas agrícolas a través del sistema SIGPAC, ofreciendo información cartográfica y parcelaria oficial. Se visualizan todas las parcelas registradas en el MAPA.

  - *Extensiones:*

    - **Guardar parcela:** Permite al agricultor registrado seleccionar y guardar una parcela específica del mapa SIGPAC para su seguimiento o gestión dentro del sistema de AgroFind. Además, si una parcela ya es del usuario o pertenece a otro agricultor, se le hará saber mediante un aviso por pantalla.

    - **Identificar parcela:** Facilita la identificación de parcelas en el mapa SIGPAC mostrando detalles adicionales al seleccionar una de ellas.

- **Consultar tiempo:**

  - Permite al agricultor obtener información meteorológica relevante para apoyar la toma de decisiones agrícolas, tanto en su ubicación actual como en cada una de sus parcelas. Estas aparecen en unas cajas con el icono y la temperatura correspondiente.

  - Extiende a *Ver tiempo en ubicación*: Si el agricultor pulsa *buscar ubicación*, el sistema muestra el tiempo actual para una ubicación geográfica específica, que podría ser una de sus parcelas o una ubicación de interés.

- **Cambiar tema:** Permite al agricultor personalizar la apariencia visual de la aplicación, eligiendo entre tema claro y oscuro.

- **Cerrar sesión:** Finaliza la sesión activa del agricultor en el sistema, asegurando la privacidad y seguridad de su información.

[69] Las condiciones están definidas por el MAPA en este enlace: https://sigpac-hubcloud.es/html/listCod/listados/usos-sigpac.html

### 5.1.4. Modelo de dominio

El modelo de dominio refleja el tratamiento de los datos de la aplicación., capturando la esencia de los datos que maneja la aplicación. Es independiente de la tecnología de persistencia subyacente (DynamoDB en este caso) y se centra en la lógica de negocio.

## Diagrama de modelo de dominio

A continuación se explica el modelo de dominio utilizando un diagrama UML (figura 5.3), para hacer más visual el mismo. No es complejo, debido a que lo complejo de la aplicación no es necesariamente la base de datos, sino la gestión de todo lo que integra, y por tanto no se ha complicado más el mismo. Se podría haber añadido un `tipo` de evento, pero al tener una base de datos tan flexible como lo que se explica posteriormente, no es necesario hacer estas clases auxiliares.

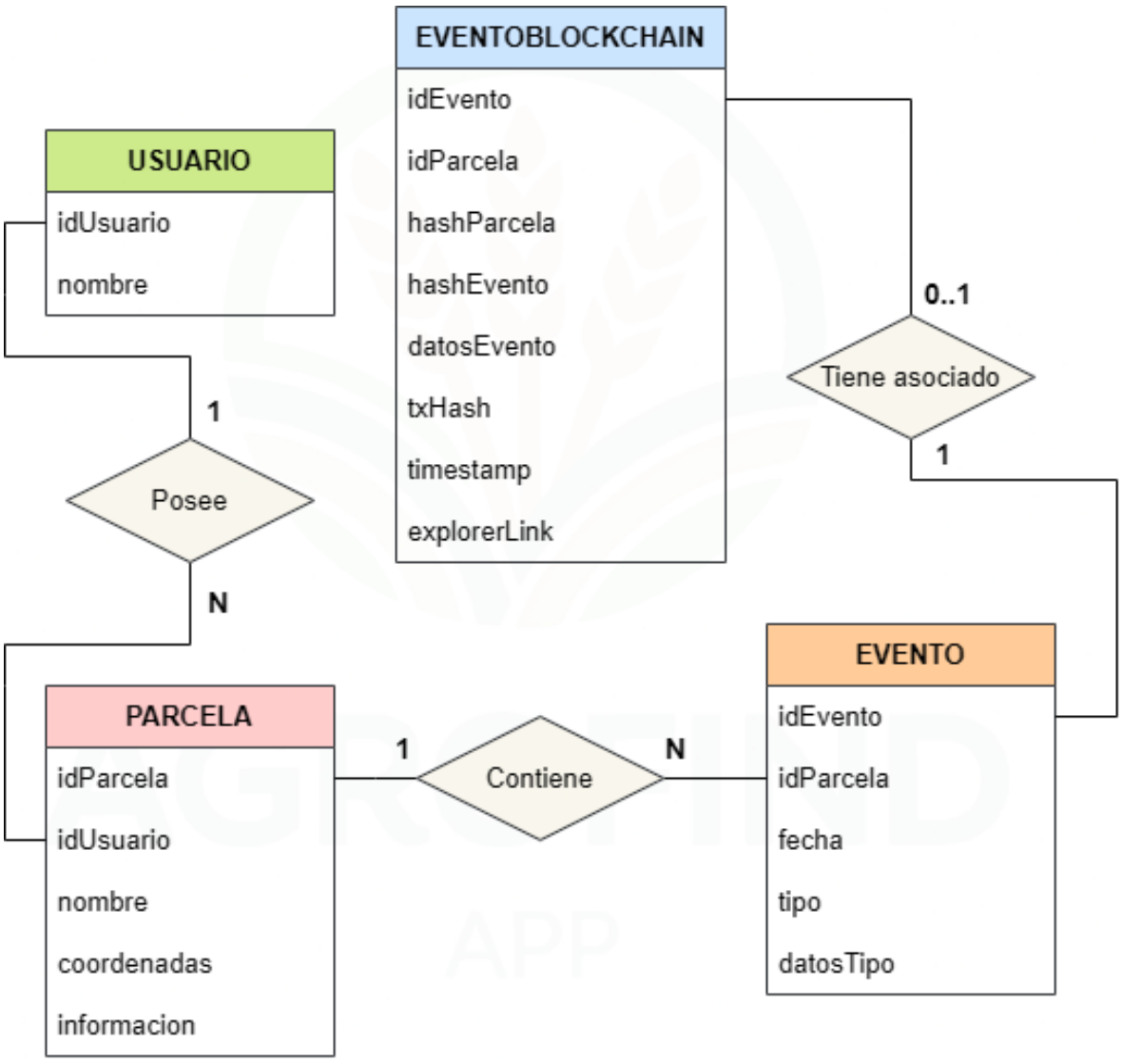



**Figura 5.3:** *Diagrama de modelo de dominio.*

### Usuario

Esta entidad representa al actor que interactúa con la aplicación, que es un agricultor registrado. Sus datos de autenticación y autorización son gestionados externamente, y se utiliza su identificador y nombre para establecer relaciones con las parcelas y eventos, pero no almacena en ningún caso ni correo electrónico ni contraseña, asegurando la privacidad.

- `idUsuario`: String (Identificador único, utilizado como clave foránea en otras entidades).
- `nombre`: String (Nombre y apellidos del agricultor).

### Parcela

La entidad `Parcela` representa una extensión de terreno agrícola gestionada por un agricultor dentro del sistema también conocida como *recinto* en caso de no tener tierra cultivable. Contiene la información necesaria:

- `idParcela`: String (Identificador único de la parcela).
- `idUsuario`: String (Referencia al `idUsuario` del agricultor que posee la parcela).
- `nombre`: String (Nombre descriptivo o identificativo de la parcela).
- `coordenadas`: List (Lista de coordenadas que conforman el polígono correspondiente a la parcela).
- `informacion`: Map (Campo flexible para almacenar detalles adicionales específicos, de tipo clave-valor).

### Evento

Un `Evento` representa un suceso o actividad registrada que ocurre en una `Parcela` específica. Esta entidad captura los detalles de las operaciones agrícolas y otros acontecimientos importantes. Los atributos que la definen son:

- `idEvento`: String (Identificador único del evento dentro del sistema).
- `idParcela`: String (Referencia a la `idParcela` a la que pertenece el evento).
- `fecha`: Date (La fecha y hora específicas en que el evento se añade).
- `tipo`: String (Categoría del evento, como `siembra`, `cosecha`, `arado`, `riego`, etc.).
- `datosTipo`: Map (Contiene datos específicos del tipo de evento, como el cultivo sembrado o la clase de riego).

**EventoBlockchain**

La entidad `EventoBlockchain` encapsula toda la información relevante de la persistencia de un `Evento` en la blockchain con los datos necesarios:

- `idEvento`: String (Referencia al `idEvento` del sistema principal al que corresponde este registro en la blockchain).
- `idParcela`: String (Referencia a la `idParcela` a la que pertenece).
- `hashEvento`: String (El hash específico que representa el evento en la blockchain).
- `hashParcela`: String (El hash de la parcela tal como fue incluido en la transacción de la blockchain).
- `datosEvento`: Map (La representación de los datos completos del evento).
- `txHash`: String (El identificador único de la transacción en la blockchain donde se registró el evento).
- `timestamp`: Date (La marca de tiempo de la transacción en la blockchain).
- `explorerLink`: String (URL a un explorador de la blockchain donde se puede verificar la transacción públicamente).

## Relaciones entre Entidades

Las relaciones descritas a continuación son lógicas y conceptuales, reflejando cómo las entidades del dominio interactúan entre sí.

- Un `Usuario` **posee** cero o muchas `Parcelas`. Una `Parcela` **es poseída por** un único `Usuario`.
  *Relación:* Uno a Muchos (1:N) de `Usuario` a `Parcela`.
- Una `Parcela` **contiene** cero o muchos `Eventos` registrados a lo largo del tiempo. Un `Evento` **ocurre en** una `Parcela` específica.
  *Relación:* Uno a Muchos (1:N) de `Parcela` a `Evento`.
- Un `Evento` **puede tener** asociado un único `EventoBlockchain` debido a que es un proceso asíncrono. Un `EventoBlockchain` **tiene asociado** un único `Evento`.
  *Relación:* Cero o Uno a Uno (1:0..1) de `EventoBlockchain` a `Evento`.

## 5.2. Arquitectura

Esta sección es de las más relevantes al proyecto, debido a que se condensa el flujo de la información en unos diagramas explicados a continuación. Para realizar estos, se ha utilizado el paquete de *cloud* en **draw.io**, que permite construir arquitecturas usando los estándares tanto de AWS como del resto de proveedores.

La arquitectura del proyecto se diseña como una **aplicación cliente** que interactúa directamente con varios servicios en la nube y APIs externas (véase figura 5.4), eliminando la necesidad de un servidor de aplicación propio centralizado.

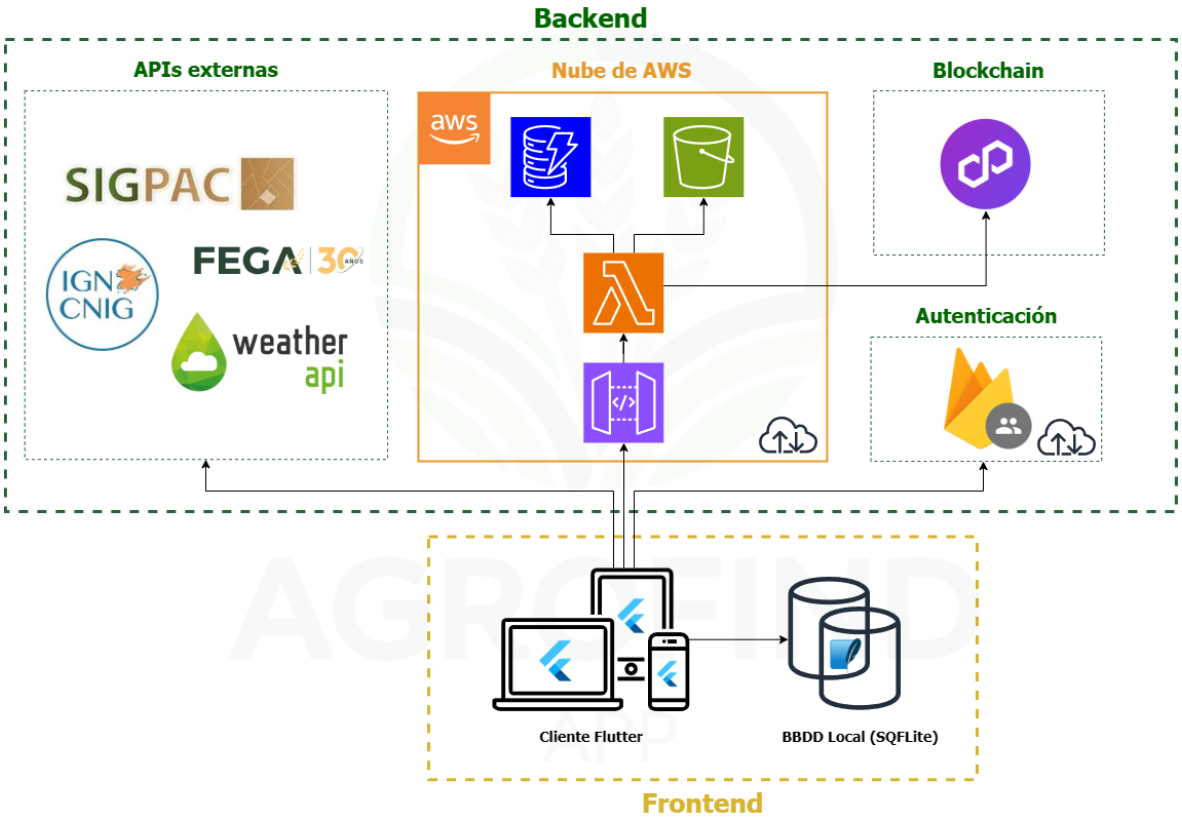



**Figura 5.4:** *Diagrama de arquitectura simplificado.*

A continuación se presenta la figura (5.5) que no es un diagrama, sino un recopilatorio de los servicios que se utilizan en AgroFind, para poder visualizarlos en un mismo lugar a modo de leyenda. No hay un diagrama completo de arquitectura porque se interactúa con muchos servicios, por tanto, se irán haciendo otros gráficos con el flujo de la información más específicos a lo largo del capítulo 6.

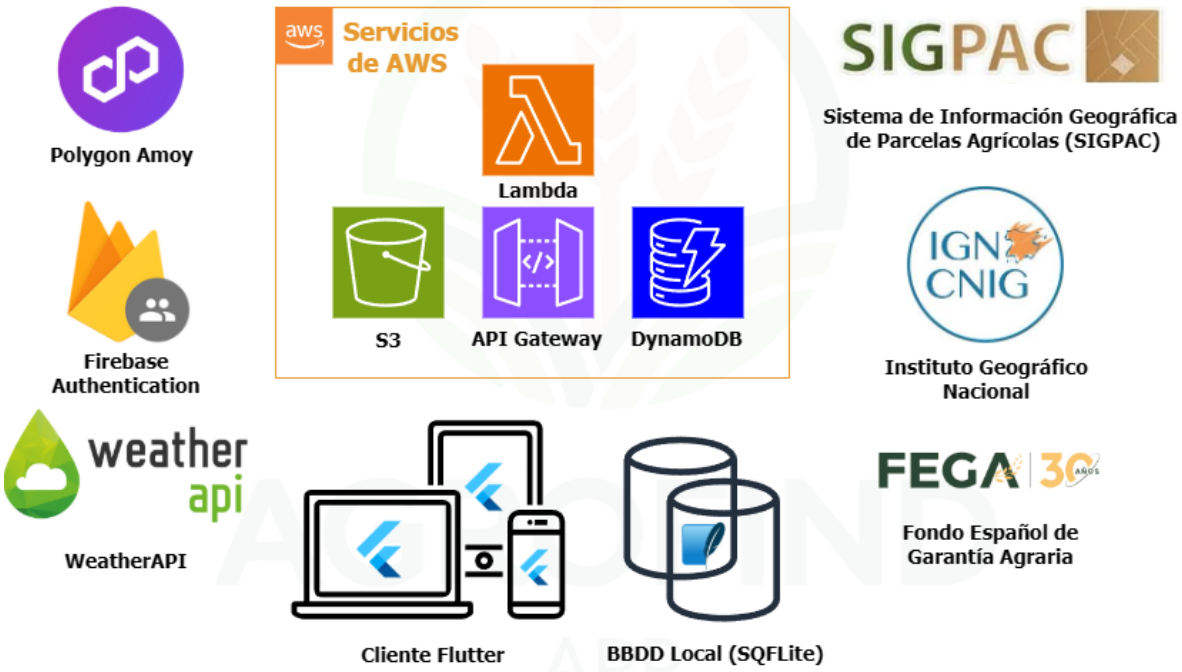



**Figura 5.5:** *Iconos de todos los servicios involucrados a modo de leyenda.*

Esta estructura se puede conceptualizar en las siguientes secciones, en las que se distingue el *Frontend* y el *Backend*, además de las bases de datos que se han empleado (estas están dentro de cada sección en base a si son locales o en la nube):

### 5.2.1. Frontend (Cliente)

Constituye la capa de interacción directa con el usuario, y de todas las comunicaciones con los servicios externos, en este caso con las varias APIs externas que se utilizan en AgroFind.

- **Cliente Flutter**: Es la aplicación de usuario final, desarrollada en Flutter y Dart. Maneja la interfaz de usuario, la lógica de presentación y es el punto de inicio para todas las peticiones a los servicios del backend, APIs externas y la autorización.
- **BBDD Local (SQLite)**: El cliente Flutter integra una base de datos local basada en SQLite. Esta base de datos se utiliza para:
  - Almacenamiento de datos offline, permitiendo a la aplicación mostrar los eventos aún sin tener conexión a internet.
  - Caché de datos para mejorar el rendimiento, reduciendo las llamadas repetitivas. Esto es muy útil en la vista de eventos y la página de ver detalles de cada parcela, donde se evitan peticiones recurrentes.

### 5.2.2. Backend

La funcionalidad tradicionalmente asociada al backend (de un servidor donde se almacena una base de datos con información y se aloja la aplicación) en AgroFind es proporcionada por varios servicios en la nube, la blockchain [70] y APIs de terceros con los que el cliente interactúa directamente. Esta es la clave para que la aplicación funcione de forma serverless, empleando estos servicios:

#### DynamoDB

Cómo se ha explicado en el capítulo 3.2.3, DynamoDB es una base de datos NoSQL y *serverless*, que se utiliza para almacenar datos no estructurados y semi estructurados de la aplicación, los eventos y las parcelas, así como los eventos que se guardan en la blockchain. El acceso a DynamoDB se realiza a través de las funciones Lambda, a las que se le llama desde la API.

En el caso de AgroFind esta información se ha gestionado mediante 3 tablas separadas entre sí. Cada una de ellas tiene una **partition key** o clave primaria y una **sort key** o clave de ordenación (véase [84]), donde la partition key es el **idUsuario** para la tabla `parcelas`, el **idParcela** para los `eventos`, y el **idEvento** para la tabla `EventosBlockchain`.

Utilizar como base de datos en la nube unas tablas en DynamoDB permite flexibilidad y no tener que preocuparse por tener un servidor, tenerlo que *hostear* o alojar en algún otro servicio. Cumple la función de una base de datos, de forma escalable y sin preocupaciones.

#### Lambda

El servicio de AWS Lambda se utiliza para ejecutar código de backend sin servidor. La aplicación invoca estas funciones a través de llamadas a la *API Gateway* para realizar una gran parte de la lógica de negocio e interacciones con las tablas de *DynamoDB* sin tener que tener un servidor que las ejecute. A continuación se listan brevemente las funciones más importantes, que posteriormente se detallarán técnicamente en la sección 6.2.2.

- **RelayerAgroFind**. Actúa como el relayer (explicación en la figura 5.11) para registrar eventos en la blockchain, generando hashes y firmando las transacciones con una clave privada almacenada en SSM (Secret System Manager, véase 6.2.2).
- **GuardarEventoBlockchain**. Se encarga de almacenar lo obtenido por la blockchain en la tabla *eventosBlockchain* de DynamoDB y después, si el evento es de tipo *cosecha*, invoca a **HacerPDFAgroFind**, que genera un certificado en formato PDF para el evento de cosecha, incluyendo un código QR con un enlace al explorador de blockchain, y lo almacena en un bucket de S3 (véase figura 6.43).

[70] Nótese que la blockchain se considera como backend para este trabajo, aunque algunos autores piensan que habría que separarla como una tercera función. Fuente: https://forum.polkadot.network/t/blockchain-is-not-a-backend-let-s-discuss/10763

- **GuardarParcelaEnDynamo**. Guarda información de las parcelas en la tabla DynamoDB *parcelas*, incluyendo coordenadas, nombre, y datos SIGPAC. De la misma manera, **GuardarEventoEnDynamo** hace la misma funcionalidad pero para los eventos individuales en la tabla *eventos*.
- **ObtenerEventosDynamo**. Recupera los eventos almacenados en DynamoDB para un usuario específico, utilizando un índice secundario (Global Secondary Index o GSI) basado en el *idUsuario*. Al igual que esta, se encuentra **ObtenerParcelasDynamo**, que recupera todas las parcelas almacenadas en la tabla DynamoDB *parcelas*.
- **CambiarNombreParcela**. Actualiza el nombre de una parcela específica en la tabla DynamoDB *parcelas*.

### API Gateway (AgroFindAPI)

Una API (Interfaz de Programación de Aplicaciones) es un conjunto de reglas que permite que diferentes sistemas de software se comuniquen entre sí. En este caso, API Gateway actúa como un **punto de entrada unificado** y seguro en el que el cliente Flutter se conecta a través de este, en el que se han definido distintas rutas para acceder a ejecutar Lambdas que hagan procesamiento, devuelvan información o añadan un evento a la blockchain.

En esencia, se crea AgroFindAPI, como una nueva API en la consola de AWS que está personalizada cómo se verá en la sección 6.2.2, y que es pieza clave en el flujo de la información, porque hace de **proxy inverso** (véase figura 5.6). Esto quiere decir que reenvía peticiones del cliente a otros servicios, cómo en este caso son las Lambdas, que se expondrán a través de distintos endpoints.

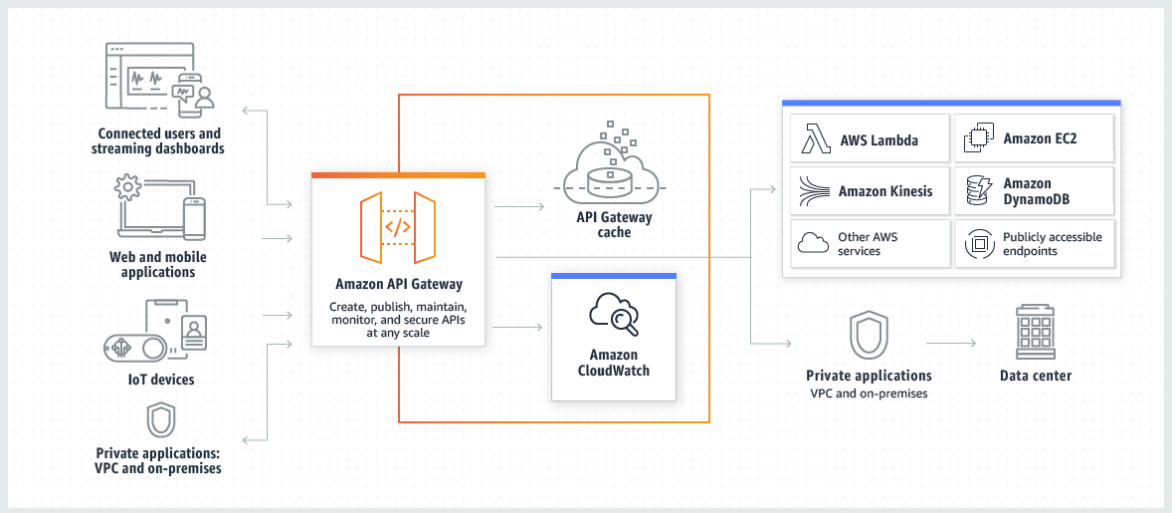
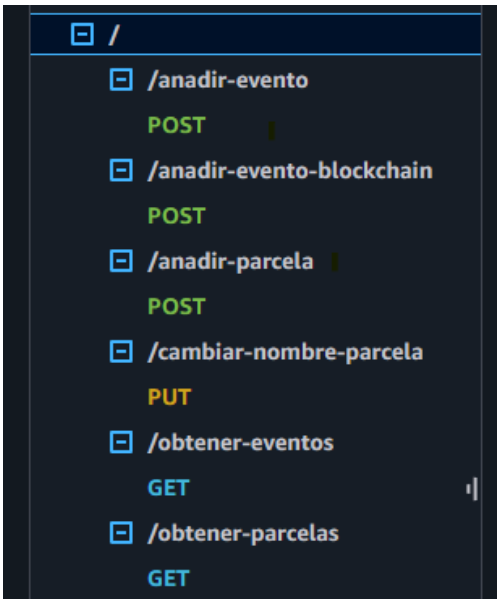



**Figura 5.6:** *Diagrama de arquitectura de API Gateway. Fuente:* https://docs.aws.amazon.com/es_es/apigateway/latest/developerguide/welcome.html

También hay que configurar los recursos de cada ruta (figura 5.7), donde cada endpoint expone una Lambda a través de la API y un método específico. En el caso de AgroFindAPI, se define que es necesaria tener la **api key** o clave de api para acceder, puesto que es un recurso público, y si no hubiera autenticación, existiría riesgo de un ataque `DoS` [71] a la app y de esta manera es AWS quién lo gestionaría con su protección base.




- **POST** `/anadir-evento`
  accede a `GuardarEventoEnDynamo`.
- **POST** `/anadir-evento-blockchain`
  accede a `RelayerAgroFind`.
- **POST** `/anadir-parcela`
  accede a `GuardarParcelaEnDynamo`.
- **PUT** `/cambiar-nombre-parcela`
  accede a `CambiarNombreParcela`.
- **GET** `/obtener-eventos`
  accede a `ObtenerEventosDynamo`.
- **GET** `/obtener-parcelas`
  accede a `ObtenerParcelasDynamo`.

**Figura 5.7:** *Rutas en API Gateway.*

En la sección de implementación (véase 6.2.2) se analizará como se ha hecho el despliegue de la API, cómo funciona la integración con las distintas Lambdas y la configuración en mayor profundidad.

### Resto de servicios de AWS

Se explicarán más adelante en detalle, entre los que se encuentran **S3** para almacenar tanto copias de seguridad, los informes generados y poder acceder a ellos de manera pública y otros tres que no se han mencionado previamente y se incluyen sus iconos correspondientes en orden en las siguientes figuras, **CloudWatch** (monitorización), **SSM** (almacenamiento de claves) e **IAM** (autorización de acceso).



[71] Un ataque de denegación de servicio (DoS) es un intento malintencionado de interrumpir el tráfico de un servidor, servicio o red, sobrecargando su infraestructura asociada con una avalancha de tráfico.

### Firebase Authentication

Es un servicio gestionado de Google que proporciona capacidades robustas de autenticación que forma parte de todos los productos que ofrece **Firebase**. AgroFind lo utiliza para el registro de usuarios, inicio de sesión, gestión de sesiones y para cuando a un agricultor se le olvida la contraseña, **delegando la complejidad de la seguridad de la identidad**. De esta manera, no se almacena en la aplicación ningún dato sensible en las bases de datos de la propia aplicación, sino que se utiliza el nombre del agricultor y un identificador de usuario que no se puede trazar ni a un correo electrónico ni a una contraseña. Este identificador es único y no se puede cambiar, lo que lo hace perfecto para usarlo como clave foránea en tablas tanto de DynamoDB cómo de SQFLite.

### Otras APIs

- *WeatherAPI*: Una API de terceros dedicada a proporcionar datos meteorológicos (temperatura, precipitaciones, pronósticos, etc.) que la aplicación consume para dar el pronóstico del tiempo en la ubicación actual, en las parcelas del usuario y permite consultar el tiempo en cualquier otro punto del mundo.
- *SIGPAC*: La aplicación interactúa con dos APIs de este sistema geográfico oficial para obtener datos detallados sobre parcelas agrícolas, entre los que se encuentran su ubicación, uso geográfico, hectáreas y más características. Lo más destacable es el poder visualizar todas las parcelas con la imagen aérea, donde cada parcela está delimitada perfectamente acorde a lo que está especificado en el *catastro nacional*. Esto diferencia AgroFind de otras herramientas, ya que hace que las parcelas sean mucho más sencillas de añadir.
- *IGN / CNIG* [72]: Se utilizan servicios cartográficos y de datos geográficos de esta institución para cargar el mapa más preciso posible y que represente la realidad en el campo, con el propósito de superponer las imagenes de SIGPAC y que tenga la mayor precisión posible.
- *FEGA*: Al igual que con SIGPAC, se accede a una API que ofrece el FEGA para obtener información sobre si una parcela es cultivable o no, además de otra información valiosa como posibles avisos relevantes al terreno (figuras 5.8 y 5.9) que puedan afectar al rendimiento del mismo.

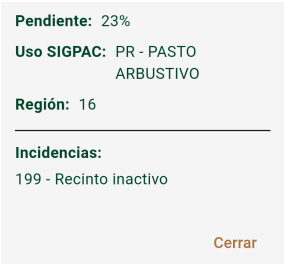



**Figura 5.8:** *Incidencia: Recinto inactivo.*

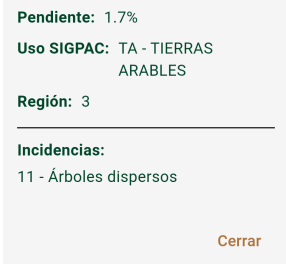



**Figura 5.9:** *Incidencia: Árboles dispersos.*

[72]IGN (Instituto Geográfico Nacional) / CNIG (Centro Nacional de Información Geográfica).

### 5.2.3. Blockchain

Esta sección es la que da cohesión a todo el proyecto, debido a que es donde reside la solución de la trazabilidad.

Cómo se ha explicado en el capítulo 3.3, la elección de Polygon no es casual, y aquí se explicará en detalle qué se ha utilizado de la misma, y se separa lógicamente del bloque de *backend* para darle más importancia, pero en la práctica, la cadena de bloques forma parte del *backend*.

Se interactúa con la testnet (red de pruebas) de Polygon, en la que está alojado el contrato inteligente que permite almacenar el histórico de los eventos en la blockchain. La aplicación no interactúa directamente con ello, sino que se hace a través de una Lambda en AWS, por tanto, cualquier uso de esta Lambda está muy controlado.

Una ventaja que tiene desplegar el smart contract en Amoy es que se hace con el mismo código que si se quisiera utilizar en Ethereum y otras blockchains basadas en la EVM (véase sección 3.3) porqué está programado en Solidity, y solo habría que cambiar la dirección del contrato en la Lambda para que siguiese funcionando de la misma manera. Indirectamente aquí habría que mencionar también las *faucets* explicadas anteriormente, porque son las que han permitido obtener $POL de prueba para desplegar todo en la *testnet*.

#### Smart Contract

Primeramente, se ha encontrado una documentación excelente para poder realizar tanto el código como las pruebas y el posterior despliegue, y esta es la realizada por *Magic Labs* [85]. A raíz de ésta se ha codificado primero el contrato inteligente usando **Remix** como IDE, ya que permite realizar muchas más tareas y no solo la codificación.

El **testing** se realiza directamente en el entorno mediante la pestaña *Deploy and Run Transactions*, donde se pueden ejecutar funciones del contrato utilizando cuentas simuladas o reales. El despliegue se realiza seleccionando un entorno como **Metamask** o similar y enviando el contrato a la blockchain configurada, que en este caso es Polygon Amoy, siguiendo las directrices de Polygon [86].

Además, esto requiere de configuración de Metamask para permitir las *meta - transacciones* o transacciones mediante un relayer [73], y para ello se hace una billetera nueva en Metamask, se copian las claves y se almacenan de forma segura en SSM. Después hay que obtener saldo para la billetera, y al estar en una red de prueba como es Polygon Amoy, hay que usar **faucets** (véase 3.3).

Una vez hecho esto, hay que codificar el contrato y desplegarlo en la blockchain con la misma billetera, para que se certifique que esa es la que se está usando.

[73] La función de las meta-transacciones se explica en la página oficial de Metamask. Fuente: [87]

El relayer y el contrato inteligente se explicarán más en detalle en siguientes secciones (5.2.3 y 6.2.3 respectivamente), pero para mayor claridad visual, se adjunta la figura 5.10 que es el contrato verificado en la blockchain [74].

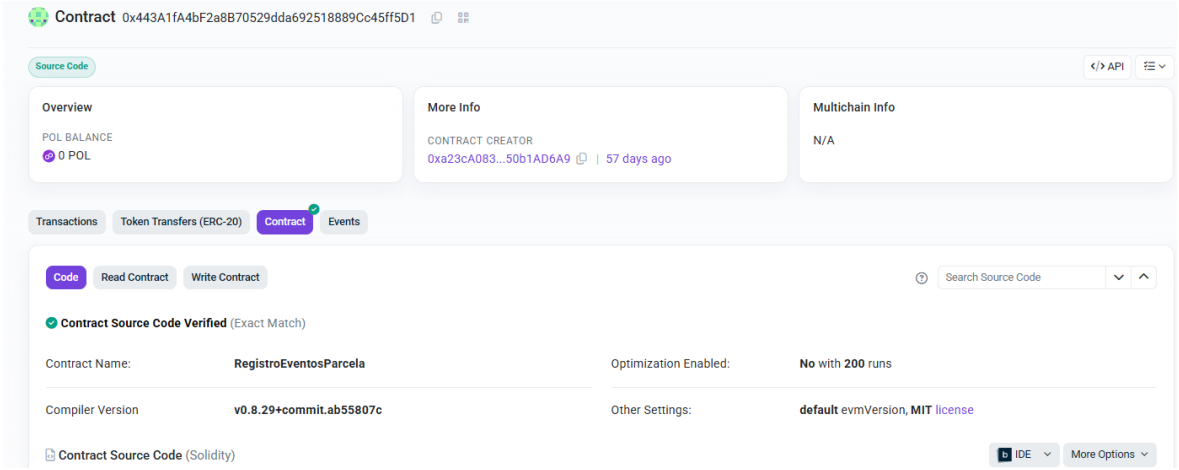



**Figura 5.10:** *Contrato RegistroEventosParcela verificado en la blockchain.*

### Uso del Relayer

Esto es una pieza clave del proyecto, ya que facilita el registro de eventos en la blockchain sin que los agricultores necesiten crear cuentas o gestionar sus propias claves privadas (figura 5.11). Simplifica la experiencia del usuario, permitiendo que los eventos se registren sin requerir conocimientos técnicos avanzados por parte de los usuarios.

Básicamente, un *relayer* es un **intermediario que actúa como puente entre un sistema externo (como una API o una app) y una blockchain** [88]. Su función principal es enviar transacciones a la blockchain en nombre de un usuario o sistema. Los *relayers* son útiles cuando se desea mantener las claves privadas seguras y se necesita automatizar interacciones con la blockchain (véase [89]) cómo es el caso de AgroFind.

En el caso de la lambda **RelayerAgroFind** (véase extracto de código 6.12), es un archivo en *Node* hecho para registrar eventos agrícolas en un contrato inteligente de la blockchain de Polygon. Este relayer recibe datos de un evento a través de una solicitud REST a través de AgroFindAPI, genera los hashes necesarios, firma la transacción con una clave privada almacenada de forma segura en SSM, y llama a la función `registrarEvento` del smart contract.

[74] RegistroEventosParcela ha sido verificado, y actualmente se puede visualizar en: https://amoy.polygonscan.com/address/0x443a1fa4bf2a8b70529dda692518889cc45ff5d1

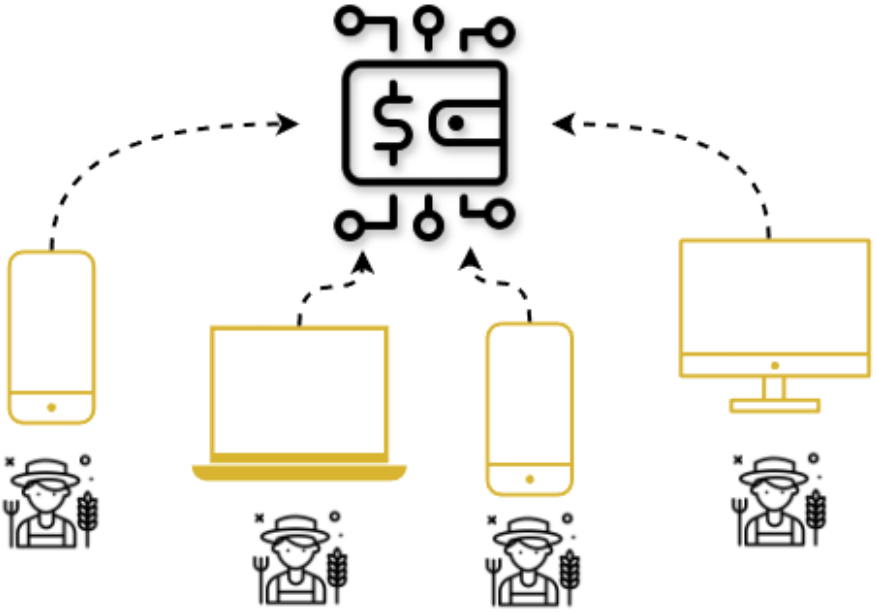


**Figura 5.11:** *Ilustración de funcionamiento de relayer.*

Las ventajas de este enfoque utilizando relayer son las siguientes:

- **Seguridad**. La clave privada está protegida y no se expone directamente.
- **Simplicidad**. Los agricultores no tienen que crear billeteras ni añadir saldo, y en caso de tener que usar más saldo, se gestiona todo en una billetera única.
- **Automatización**. Permite registrar eventos sin intervención manual.

### 5.2.4. Conexión de Cliente a Backend

Esta sección analizará, utilizando diagramas de flujo, cómo se mueve la información entre la aplicación y el backend (incluyendo la blockchain).

Se mencionarán estos durante la sección de implementación y por tanto es importante comprender las peticiones a la AgroFindAPI que posteriormente llama a las Lambdas, cómo se almacena la información en DynamoDB y cuándo se crea un nuevo evento utilizando el smart contract.

Primeramente, la manera de guardar tanto las parcelas (figura 5.12) cómo los eventos (figura 5.13) es idéntica. Se definirán dos *endpoints* o puntos finales en la API, cada uno con una ruta distinta, a los que se llamará utilizando una consulta POST con la información relevante en cada caso. La API llamará a cada una de las Lambdas y solo añadirá la nueva instancia en caso de que no existiera ya en la tabla previamente.

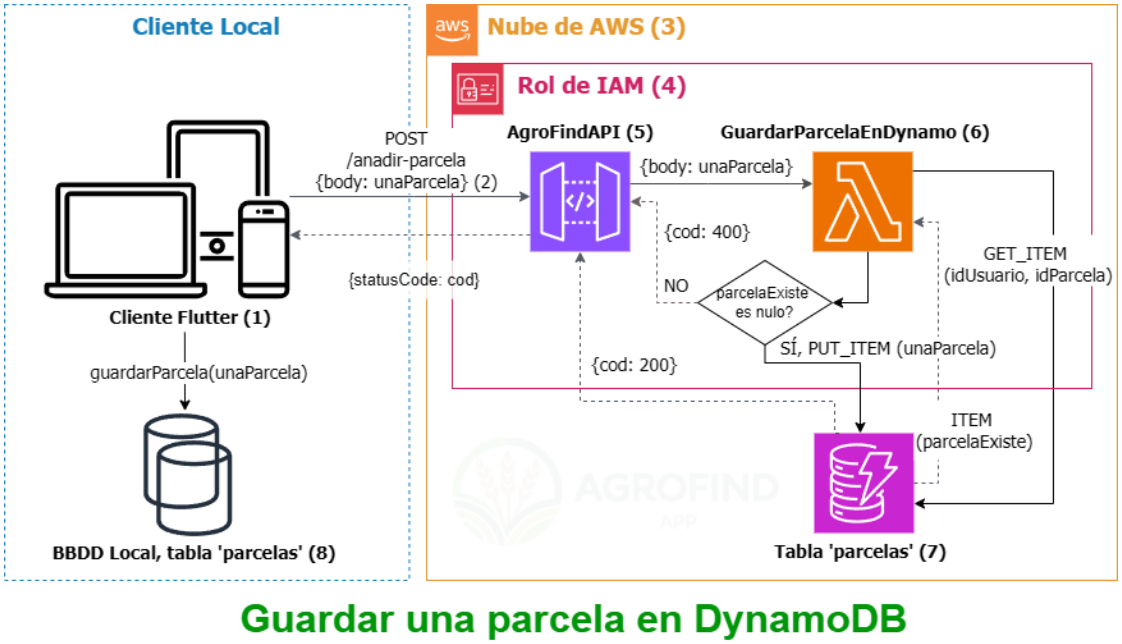



**Figura 5.12:** *Diagrama de flujo para guardar una parcela.*

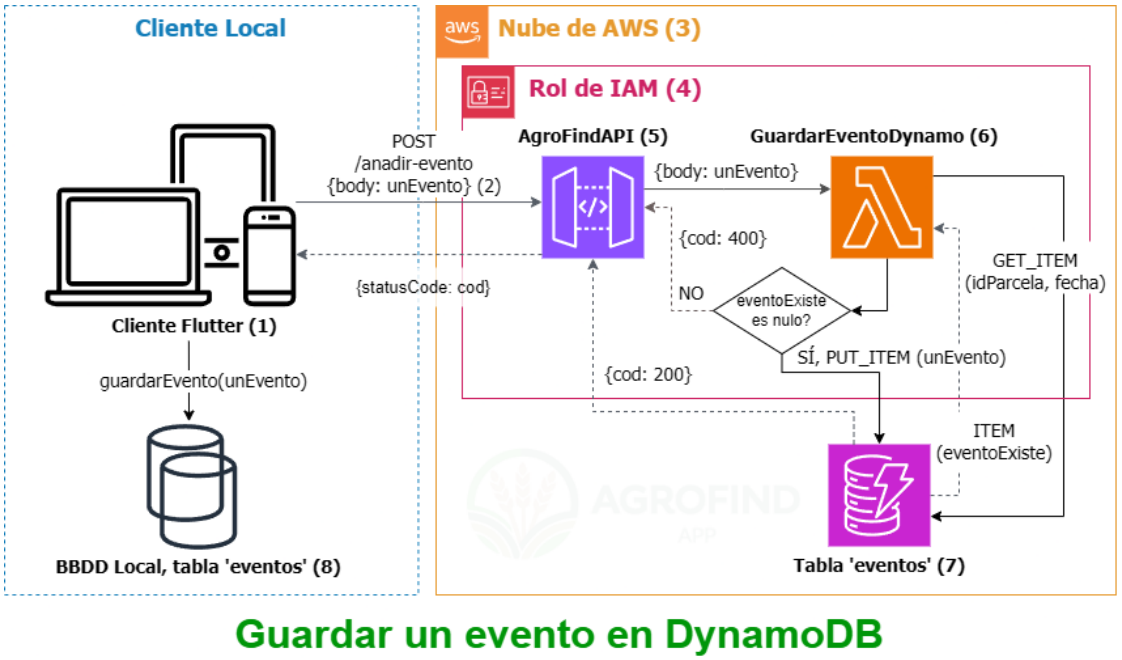



**Figura 5.13:** *Diagrama de flujo para guardar un evento.*

Por otra parte, para obtener parcelas (figura 5.14) y eventos (figura 5.15) cambia un poco el flujo. Esto se debe a que en la tabla de eventos se ha creado un GSI, que es una estructura que permite realizar consultas eficientes sobre atributos que no forman parte de la clave primaria de la tabla, manteniéndose automáticamente sincronizado con los datos originales (véase [90]). De esta manera, se obtienen consultas mucho más rápidas y se puede filtrar por otros atributos como el idUsuario.

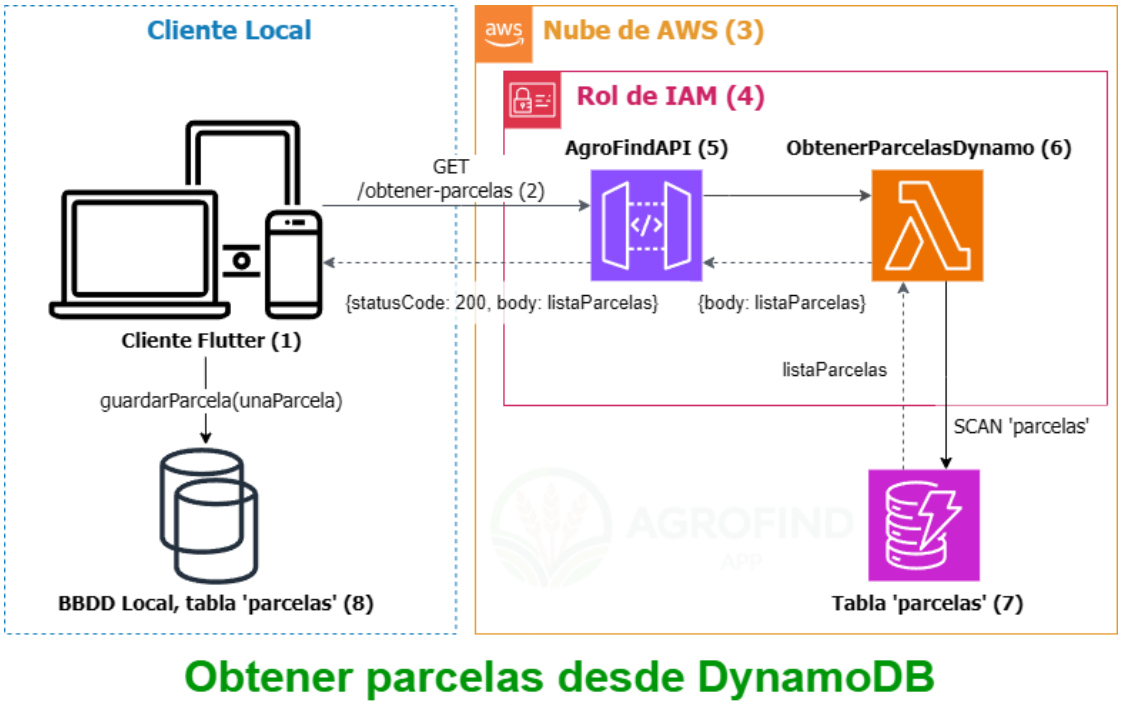



**Figura 5.14:** *Diagrama de flujo para cargar las parcelas.*

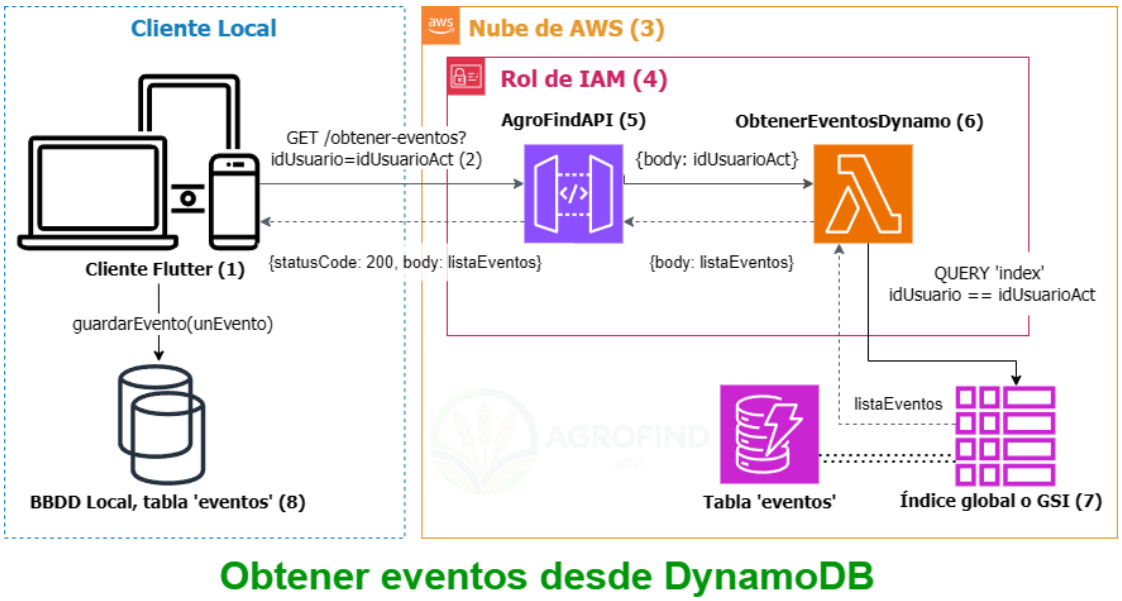
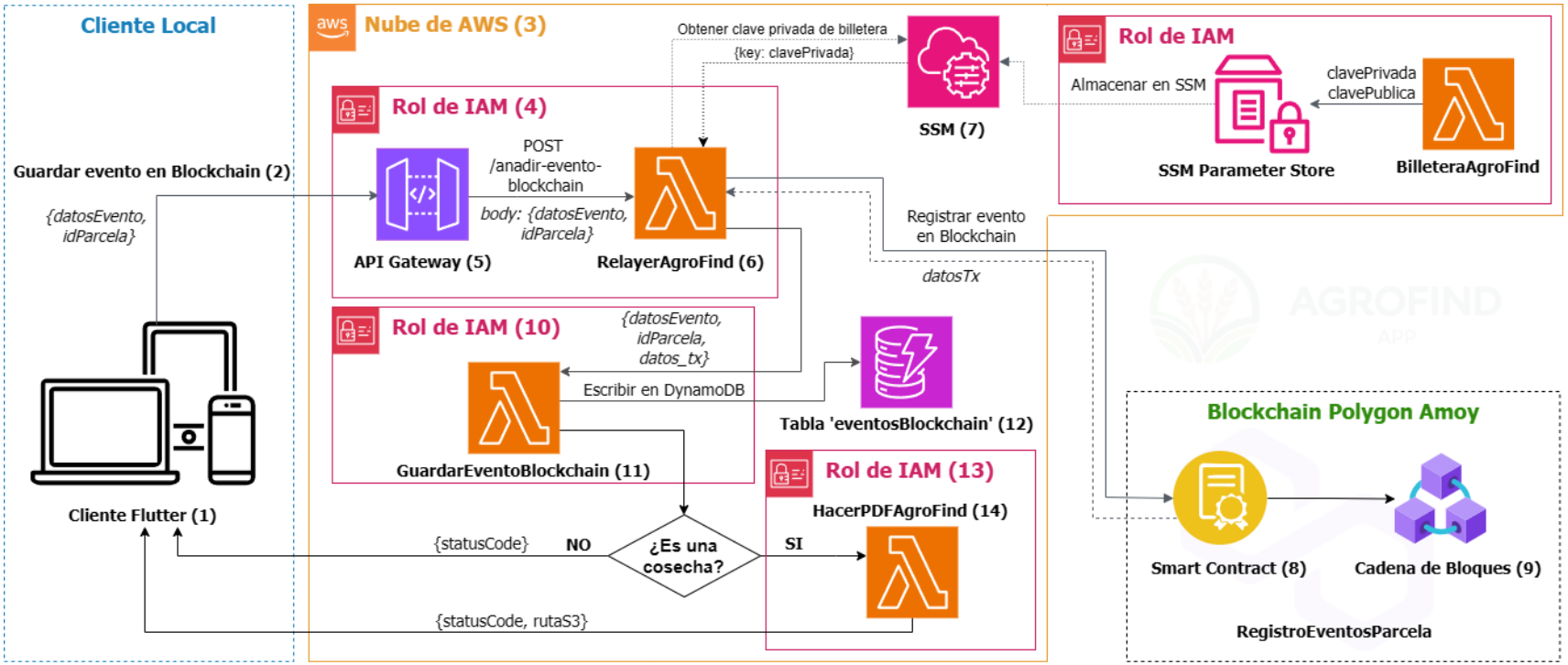



**Figura 5.15:** *Diagrama de flujo para cargar los eventos.*

El siguiente diagrama explica cómo se añade un nuevo evento a la blockchain y posteriormente a la tabla `eventosBlockchain` de DynamoDB. Cada vez que se añade un nuevo evento se hacen dos peticiones simultáneas, la que añade el evento a la tabla `eventos` (figura 5.13, ya analizada), y esta, que es la más compleja.

Una vez hecha la petición al *endpoint* de `/anadir-evento-blockchain`, se invoca al *relayer*, que se encarga de ejecutar el smart contract, y de posteriormente invocar a otra Lambda que guarda el eventoBlockchain en la tabla. Además, en caso de que sea una cosecha, se crea un PDF en una ruta concreta, que es a donde apunta el código QR que se muestra en el cliente (véase 6.2.2).




**Añadir nuevo evento a la blockchain y a DynamoDB**

**Figura 5.16:** *Diagrama de flujo para añadir un nuevo evento a la blockchain y almacenarlo en DynamoDB.*

Finalmente, cuando se cambia el nombre a una parcela el flujo es muy directo, ya que se hace la petición correspondiente que invoca a la lambda `CambiarNombreParcela` y esta ejecuta una función `Update` en la tabla parcelas, dónde actualiza el nuevo nombre a la parcela correspondiente. Nótese que no se comprueba si la parcela existe, esto se debe a que se comprueba primero localmente y por tanto nunca se va a dar la situación de que una parcela esté en local y no en la tabla en la nube.

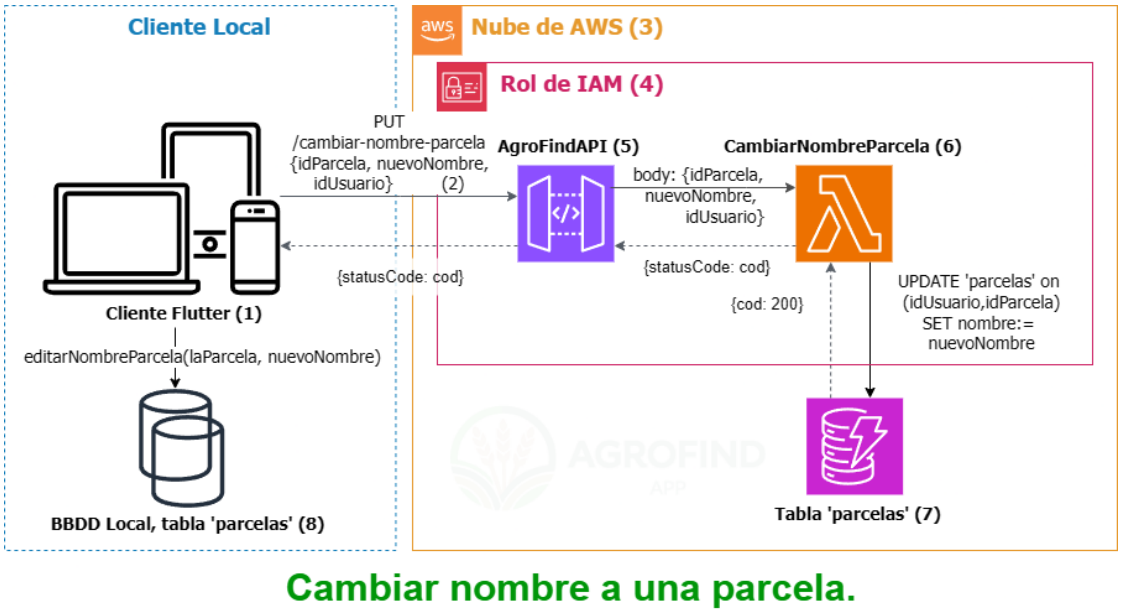



**Figura 5.17:** *Diagrama de flujo para cambiar el nombre a una parcela ya existente.*

Esta arquitectura distribuida y basada en servicios permite a AgroFind ser ágil, escalable y resiliente, aprovechando las capacidades de proveedores de nube para gestionar la infraestructura y la lógica de backend, siempre manteniéndose serverless que es uno de los objetivos principales del trabajo.

# 6. Implementación

Habiendo definido en las secciones anteriores todo lo necesario para el desarrollo de la aplicación, en esta sección se hará una explicación exhaustiva sobre cómo se ha implementado todo el sistema, desde la parte de interfaz de usuario hasta todo el backend.

## 6.1. Cliente Flutter

Antes de introducir los detalles del código, es necesario explicar brevemente cómo poder configurar el entorno de desarrollo, para replicarlo si fuera necesario en otro dispositivo.

### 6.1.1. Entorno de desarrollo

Para realizar toda la codificación del frontend, la herramienta principal que se ha usado ha sido **Visual Studio Code** o VSCode, y por tanto es necesario explicar cómo se ha configurado este IDE.

Inicialmente, hay que instalarlo desde la página oficial [75]. En el caso del autor, esto se ha hecho íntegramente en Windows, por tanto se ha decidido descargar la versión de Windows 10. Lo segundo a tener en cuenta son los *plugins* o extensiones, que son la razón principal de haber elegido esta IDE. VSCode cuenta con una infinidad de estos, que permiten tanto personalizar el aspecto visual de la aplicación (el tema) como añadir nuevas funcionalidades a las ya existentes.

Los plugins absolutamente esenciales son **Flutter**, **Dart** y **Flutter Widget Snippets**, donde este último ayuda a programar autocompletando estructuras comunes al escribir patrones: `stf + enter` crea un *StatefulWidget*, mientras que `stf + enter` crea un *StatelessWidget*, además de otros tantos hotkeys o atajos que ofrece, esto ayuda a tener más velocidad a la hora de codificar.

Al haber realizado el proyecto con control de versiones, las extensiones de **Git** y **Github** son necesarias, y se ha utilizado **GitLens**, que facilita a la estructura de ramas y ayuda a unificarlas desde una interfaz. Además, cómo se ha utilizado Copilot para la asistencia en la programación, el plugin de **Github Copilot** ha sido de gran utilidad.

[75] Enlace a la descarga de VSCode: https://code.visualstudio.com/

Respecto a otros aspectos que no sean plugins, para poder utilizar el entorno para poder probar la aplicación en Android, Linux, Windows y Web [76] (se omiten iOS y macOS por el momento, se explicarán las razones más adelante), es necesario tener la SDK de Android instalada. La manera más sencilla es descargar **Android Studio** [91], aunque no se vaya a usar esta IDE para el desarrollo. Al descargarlo es necesario marcar las casillas *Android SDK, Android SDK Platform y Android Virtual Device*, para poder también crear un emulador para probar la aplicación. Este se crea directamente desde Android Studio, usando la herramienta de *Device Manager*.

También hay que instalar el SDK de Flutter, que se obtiene desde el tutorial que ofrece Flutter para instalarlo en Windows [92]. Una vez hecho esto, para comprobar que la instalación es correcta, es conveniente ejecutar `flutter doctor`, comando que comprueba que Flutter esté instalado correctamente.

A continuación hay que configurar Firebase. Primero es necesario instalar la interfaz de línea de comandos (CLI) de Firebase, y para ello hay que acudir a su página oficial [93] y seguir el tutorial que esta proporciona. Después, instalar la CLI de manera global para Dart y añadir las dependencias correspondientes:

```
dart pub global activate flutterfire_cli
flutter pub add firebase_core
flutter pub add firebase_auth
```

Finalmente, habría que configurar el proyecto en la web de Firebase (véase 6.2.1). Una vez realizados todos los pasos, ya se podría empezar a programar en el entorno. Para ejecutar la aplicación, se clona desde el repositorio de Github, después se accede a la carpeta correspondiente, se arranca el emulador si se va a usar (si no se usa y hay un dispositivo conectado no hay que hacer este paso), y se ejecuta la aplicación:

```
git clone <enlace_al_repositorio>
cd agrofind/
flutter emulators --launch <nombre_del_dispositivo> (opcional)
flutter run
```

Para aprovechar el *hot reload* (véase 3.1.1) de Flutter, es tan sencillo como pulsar el botón de recarga (r) en la terminal mientras se está ejecutando la aplicación en el entorno.

### 6.1.2. Jerarquía de clases

A continuación se presenta la jerarquía de clases (figura 6.1), en el que se agrupan las mismas por categoría. Se explicarán individualmente, donde están: Utilidades, Páginas, Componentes, Modelos, Temas, Base de Datos, Autorización, SIGPAC y Main. Las clases se representan con un rectángulo y las categorías están dibujadas como paquetes [77] UML.

[76] Para ejecutar una versión web, Flutter ofrece un comando, y después se puede levantar el servicio con un servidor sencillo usando Python:
`flutter build web`, y posteriormente `python -m http.server -directory build/web`

[77] Esto se denomina como diagrama de paquetes, que suele ser comúnmente utilizado para representar la estructura o jerarquía de una aplicación. Fuente: [94].

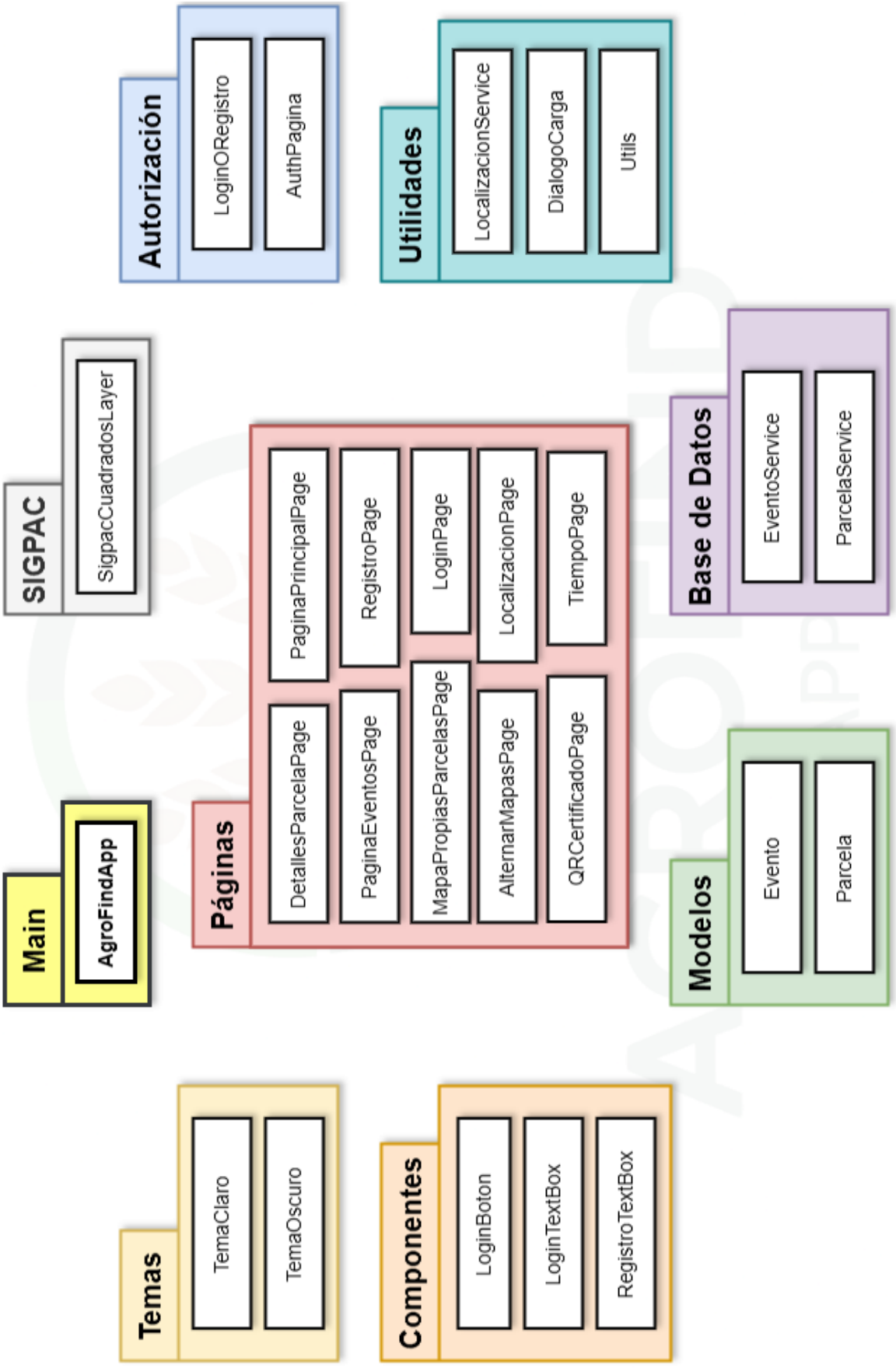



**Figura 6.1:** *Diagrama de paquetes que representa la jerarquía de clases.*

- **Utilidades**: Agrupa clases que proporcionan funcionalidades auxiliares y servicios generales. Incluye *LocalizacionService*, que gestiona servicios de ubicación, calcula distancias y determina si una parcela está en cierta localización; *DialogoCarga*, un diálogo de carga personalizado con el logo para dar más profesionalidad a la aplicación; y *Utils*, una clase con funciones útiles de varios ámbitos, desde crear una *Snackbar* personalizada hasta un mapeo con los usos SIGPAC.
- **AgroFindApp**: Representa la clase principal de la aplicación, encapsulando la lógica y el flujo general de AgroFind.
- **SigpacCuadradosLayer**: Gestiona todo lo relacionado con la carga de la información de SIGPAC. En la práctica, es una capa más del mapa, pero que se ha tenido que hacer a medida para poder convertir la información entrante a un mapa a escala.
- **Componentes**: Contiene clases que definen elementos reutilizables de la interfaz de usuario. *LoginTextBox*, un campo de texto para la entrada de credenciales; *LoginBoton*, un botón para iniciar sesión; y *RegistroTextBox*, un campo de texto para la entrada de datos de registro.
- **Páginas**: Engloba todas las clases que representan las diferentes interfaces, que se explicarán en un apartado separado junto a imágenes sobre cómo es cada una de ellas. Estas incluyen:
  - *LoginPage*: La interfaz para el inicio de sesión de agricultores que incluye el botón para recuperar contraseña.
  - *RegistroPage*: La pantalla para el registro de nuevos usuarios.
  - *PaginaPrincipalPage*: Gestiona la navegación entre páginas, el título y el cambio de tema en cada página del menú principal (la lista de eventos, ambos mapas y el tiempo).
  - *PaginaEventosPage*: Enseña la lista de eventos y es lo que se carga al hacer login.
  - *AlternarMapasPage*: Permite cambiar entre *LocalizacionPage* (carga el mapa SIGPAC y todas las parcelas, tanto propias como de otros usuarios) y *MapaPropiasParcelasPage* (carga las parcelas propias y permite acceder a ver detalles).
  - *TiempoPage*: Permite ver el tiempo en todas las parcelas del agricultor y en la ubicación actual. También permite especificar otra ubicación.
  - *DetallesParcelaPage*: Muestra información detallada sobre una parcela específica.
  - *QRCertificadoPage*: Se carga al haber hecho una cosecha, y muestra el código QR del certificado junto con el enlace al que lleva el código.

- **Modelos**: Define las estructuras de datos fundamentales utilizadas en la aplicación. Incluye *Evento*, que representa un suceso o actividad en una *Parcela*, que es una unidad de terreno o recinto agrícola.
- **Temas**: Agrupa clases para la gestión de la apariencia visual personalizada de la aplicación. Se subdivide en *TemaClaro*, para un esquema de colores claros, y *TemaOscuro*, para un esquema de colores oscuros.
- **Autorización**: Contiene clases relacionadas con la autenticación. Incluye *LoginORegistro*, que maneja la lógica de inicio de sesión o registro, y *AuthPagina*, componente para evitar hacer login si hay una *cookie de sesión* activa.
- **Base de Datos**: Agrupa las clases de servicio que interactúan directamente con la base de datos para la persistencia y recuperación de información. *EventoService* usa el patrón **factory** para gestionar los datos de la tabla *eventos*, mientras que *ParcelaService* hace la misma función pero para la tabla de *parcelas*.

### 6.1.3. Parcela y Evento

Antes de explicar la interfaz de usuario, es necesario analizar los dos modelos o clases de datos básicas que se utilizan en la aplicación, que son **Evento** y **Parcela**. Éstas se han explicado en la sección anterior de manera general, pero es necesario profundizar en el código y la usabilidad de cada una. Nótese que se han visto previamente en el modelo de dominio (sección 5.1.4). A partir de este punto, se referirá a la API Gateway de AWS como **AgroFindAPI**, para distinguirla de otras APIs que se utilizan en el proyecto.

#### Parcela

Se define el código en el extracto de código 6.2, desde donde la información particular de una parcela a partir de punto geográfico se obtiene haciendo llamadas al Servicio de Teselas Vectoriales (MVT) de SIGPAC Hubcloud [96].

- **`int?` id**. Se obtiene de juntar como número entero todas las **características de la parcela** que SIGPAC determina que son identificativas, que son: *provincia, municipio, agregado, zona, polígono, parcela y recinto*. Al unirlas todas en un entero largo, quedan identificadores únicos para parcela. Por ejemplo, una parcela en Medina de Pomar (Burgos) tendría el id 921440505380, mientras que una parcela en Dima (Bizkaia) tendría el id 4826001517.
- **`List<List<double>>` coordenadas**. Estas son las listas de coordenadas que forman el polígono de la parcela, cuyo formato original es en WKT (Well Known Text). Para poderlas utilizar, hay que transformarlas desde cómo se obtienen al hacer la petición al servicio *recinfobypoint* de *SIGPAC Hubcloud* al formato que se usa en AgroFind y esto se hace en la función `convertirWktACoordenadas` en el paquete `utils` siguiendo estos pasos:

1. La función extrae todos los valores numéricos del `String` tipo WKT (que representa una geometría) usando una expresión regular. Hay que tener en cuenta que el WKT usa un formato EPSG:3857 [78], mientras que flutter_map recomienda trabajar con el formato EPSG:4326 [79] porque es el qué usan los sistemas GPS [97].
2. Agrupa esos valores en pares numéricos y los transforma de EPSG:3857 a EPSG:4326 (latitud y longitud), usando el método `unproject()` [80].
3. Finalmente, construye y devuelve una lista de objetos `LatLng` que representan las coordenadas geográficas interpretables por `flutter_map`.

Matemáticamente (véase [98]) esto implica lo siguiente:

$$(x, y) \in \mathcal{R}^2, \text{ como latitud y longitud en metros, EPSG:3857}$$

$$R = 6378137m \qquad \text{(radio de la Tierra en metros)}$$

$$\lambda = \frac{x}{R} \qquad \text{(longitud en radianes)}$$

$$\phi = \frac{\pi}{2} - 2 \cdot \arctan\left(e^{-y/R}\right) \qquad \text{(latitud en radianes)}$$

$$\text{long} = \lambda \cdot \left(\frac{180}{\pi}\right) \qquad \text{(longitud en grados, EPSG:4326)}$$

$$\text{lat} = \phi \cdot \left(\frac{180}{\pi}\right) \qquad \text{(latitud en grados, EPSG:4326)}$$

La función las convierte a un tipo de dato `LatLng`, que para almacenarlo tanto en SQFLite como en DynamoDB habría que pasarlo a un tipo de dato elemental, `List<double>`, pero esto es trivial, utilizando una de las operaciones elementales de Dart (extracto 6.1).

```
/* resto de código */
Parcela laParcela = Parcela(
    id: int.parse(idParcela),
    nombre: 'Campo $idParcela',
    idUsuario: usuarioActual!.uid,
    coordenadas: coordenadas
        .map((latLng) => [latLng.latitude,
                          latLng.longitude])
        .toList(),
    informacion: data[0],
);
/* resto de código */
```

**Extracto de código 6.1:** *Creación de un nuevo objeto Parcela, clase LocalizacionPage.*

[78] El formato EPSG:3857, mejor conocido como Pseudo-Mercator está medido en metros. Fuente: https://epsg.io/3857

[79] El formato EPSG:4326 o WGS84 está medido en grados. Fuente: https://epsg.io/4326

[80] Esta función está definida en la documentación de la clase `Projection` de `flutter_map`.

- **`Map<String, dynamic>` informacion**. Este atributo contiene toda la información de la parcela que no es estrictamente necesaria, pero que el agricultor puede consultar si así lo desea. Funcionalmente actúa como si fuera un JSON, en el que se almacena cada ítem como clave-valor, donde la clave es siempre una cadena de texto, pero el valor es dinámico (véase 3.1.1). Entre esta información obtenemos el uso SIGPAC, las hectáreas, si hay alguna incidencia registrada con esa parcela o la inclinación de la misma.
- **`String?` nombre**. Es un nombre para poder identificar la parcela más fácilmente. Por defecto, el nombre de cada parcela está definido como *Campo* y el idParcela correspondiente. Una vez cambiado, ese nombre se almacena primero en DynamoDB cambiando el nombre de la parcela y después en local, donde se utilizará como la nueva cadena de caracteres al ver detalles de una parcela y en el menú de eventos.
- **`final String?` idUsuario**. Se obtiene como en el segundo ejemplo de la sección 6.2.1, en el que es un identificador único de un usuario. De esta manera, no se almacenan datos sensibles, sino que es solo el id asociado, ya que este no cambia nunca.

```
class Parcela {
    int? id;
    List<List<double>> coordenadas;
    Map<String, dynamic> informacion;
    String? nombre;
    final String? idUsuario;
/* resto de código */
```

**Extracto de código 6.2:** *Atributos de la clase Parcela.*

### Evento

Cuando se hizo el modelo de dominio, se definió el evento con un idEvento y un idParcela, que es como se almacena en DynamoDB, pero localmente, no es necesario tener el idEvento, ya que para una parcela no hay más de un evento en un instante de tiempo (extracto de código 6.3).

- **`Parcela` parcela**. Se ha añadido que el evento guarde la parcela de la que es parte, para poder acceder de forma más rápida a la vista de **ver detalles parcela**. Puede parecer información innecesaria, pero esto es conocido como redundancia [99] y ayuda a hacer más eficiente la aplicación.
- **`final int` idParcela**. De la misma manera, se almacena el id de la parcela por separado, para dar la funcionalidad de búsqueda por parcela sin tener que entrar dentro del objeto para buscarlo.
- **`final String` tipo**. Este atributo es el que define si un evento es de tipo **cosecha**, **riego**, **siembra** u otros. Determina cuando se tiene que solicitar el código QR a la AgroFindAPI, que es cuando se cosecha una parcela (si no se ha sembrado anteriormente, la cosecha no se guardará, porque no tiene sentido físico). Se obtiene cuando el agricultor lo especifica al añadir el evento.
- **`final DateTime` fecha**. La fecha en la que el evento se ha creado, se usa para fines de trazabilidad.
- **`final Map<String, dynamic>?` datosTipo**. Otro atributo de tipo clave-valor con un map dinámico, que en este caso se usa para definir los datos extra de cada tipo de evento, por ejemplo un **arado** puede ser tanto *arado de subsuelo*, como *labrado* o un *arado de superficie*.

```
class Evento {
  Parcela parcela;
  final int idParcela;
  final String tipo;
  final DateTime fecha;
  final Map<String, dynamic>? datosTipo;
/* resto de código */
```

**Extracto de código 6.3:** *Atributos de la clase Evento.*

La selección de cultivos [81] se ha realizado considerando los principales productos agrícolas sembrados en España, basándose en datos del MAPA (véase [100]). Se han priorizado aquellos cultivos con mayor presencia en el sector primario nacional, abarcando herbáceos, leguminosas, hortícolas, industriales y frutales de cáscara. Además, se ha buscado un equilibrio entre representatividad y simplicidad, evitando listas extensas que dificulten la navegación. Por último, se ha incluido la opción *no especificado* para dar flexibilidad en casos no contemplados, con la posibilidad de ampliar la lista si es necesario.

[81] Lista de cultivos: alfalfa, algodón, almendra, alubia, arroz, avena, cacahuete, calabaza, calabacín, cebada, cebolla, centeno, chufa, colza, espelta, garbanzo, girasol, guisante, judía, lechuga, lenteja, maíz, patata, pimiento, pistacho, quinoa, remolacha, soja, sorgo, trigo, tomate y zanahoria.

### 6.1.4. Interfaz de usuario (UI)

Respecto al aspecto visual de la aplicación, se hará un recorrido por la misma explicando cada interfaz, con capturas de pantalla que ilustran el contenido. Se seguirá un orden natural de la aplicación, haciendo a su vez de manual de usuario. Además, se incluirán extractos de código cuando se crea necesario para explicar alguna funcionalidad, donde todos los extractos serán una parte del código, pero siempre una pequeña porción, para ayudar a la legibilidad.

#### Main y AgroFindApp

La función principal (`main`) que da comienzo a la aplicación, es una función asíncrona o `async`. Inicialmente, carga todas las variables del entorno, principalmente las guardadas en un archivo `.env`, que son las que contienen las *API Keys* de la API Gateway (AWS) y de WeatherMap. Esto se hace utilizando la librería `flutter_dotenv`, cómo se puede ver en el extracto de código 6.9. Posteriormente se define la clase **AgroFindApp**, que usa un `ValueNotifier` para que la aplicación siempre tenga el valor del tema (claro u oscuro) actualizado y devuelve un `MaterialApp`, que es el tipo específico en Dart para dar comienzo a la ejecución de la interfaz de usuario. En el caso de AgroFind, se llama a **AuthPagina**, que se explica en la siguiente sección.

#### Autenticación

Cuando se abre la aplicación se comprueba usando **AuthPagina** si hay alguna *cookie de sesión* activa, usando el `StreamBuilder` para hacer la comprobación con la instancia de Firebase como se ve en el extracto de código 6.4.

```
/* inicio de código */
body: StreamBuilder(
    stream: FirebaseAuth.instance.authStateChanges(),
    builder: (context, snapshot) {
        // mientras se cargue el estado
        if (snapshot.connectionState == ConnectionState.waiting) {
            return const Center(child: CircularProgressIndicator());
        } else if (snapshot.hasData) {
            // en caso de haber una cookie de sesión
            return PaginaPrincipalPage(alCambiarTema: alCambiarTema);
        } else if (snapshot.hasError) {
            // si ocurre algun error
            return Center(child: Text("Error: ${snapshot.error}"));
        } else {
            // si no hay una sesión iniciada
            return LoginORegistro();
/* cierre de llaves */
```

**Extracto de código 6.4:** *Elección de que página cargar, clase AuthPagina.*

**LoginORegistro**, comprueba si es la primera vez que se abre la aplicación. En ese caso, se visualiza la página de registro **RegistroPage**, y en caso contrario se visualiza la página de inicio de sesión **LoginPage**.
Visualmente, estas dos páginas (figuras 6.2 y 6.3) tienen este aspecto:

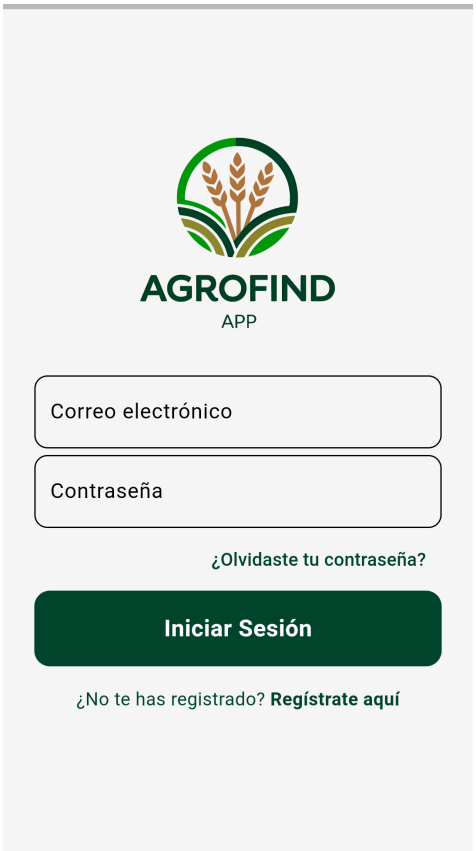



**Figura 6.2:** *Interfaz de inicio de sesión, clase LoginPage.*


**Figura 6.3:** *Interfaz de registro, clase RegistroPage.*

Ambos comparten dos elementos: un botón personalizado, que es un *GestureDetector* que ejecuta la función que se le ha pasado como parámetro al pulsarlo (extracto 6.5), y un `TextField` hecho a medida (clase `LoginTextBox`), que sirve para introducir los datos al autenticarse y que tenga el aspecto del tema de la app.

```
/* imports varios */
class BotonLogin extends StatelessWidget {
  final String texto;
  final void Function()? alPulsar;
  // parametros de entrada a la función
  const BotonLogin({super.key, required this.texto,
                    required this.alPulsar,});
  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: alPulsar,
/* resto de código */
```

**Extracto de código 6.5:** *Botón personalizado, clase BotonLogin.*

Al rellenar el formulario de registro con los datos como en la figura 6.4, la aplicación realiza la creación de la cuenta en Firebase (véase sección 6.2.1). Una vez que se ha hecho el registro correctamente, aparece un `AlertDialog` (figura 6.5) que da la bienvenida al agricultor de manera personalizada, para dar una experiencia más cercana.

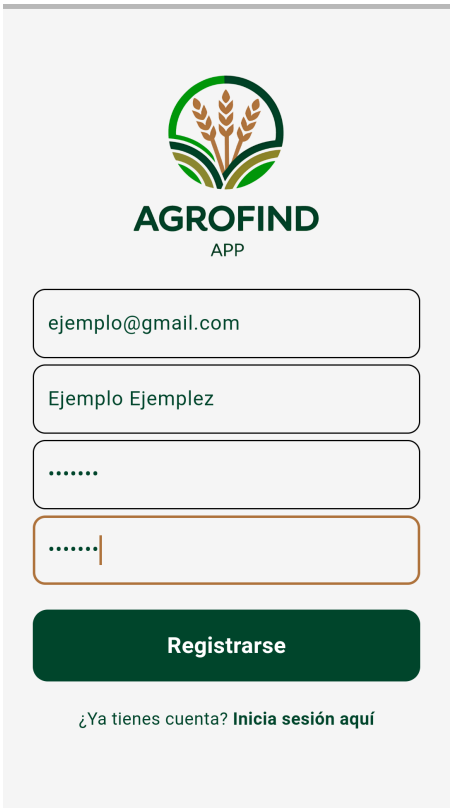



**Figura 6.4:** *Vista de registro con datos de ejemplo.*


**Figura 6.5:** *Diálogo de bienvenida.*

Respecto al inicio de sesión, funciona de una manera muy similar. La versión en modo oscuro es idéntica para ambas pantallas (figura 6.6) y para ver de forma gráfica el funcionamiento, la figura 6.7 muestra como todos los formularios son robustos, y que hacen todas las comprobaciones pertinentes para que solo se llame al servicio en caso de haber rellenado correctamente todas las `TextBox` (figura 6.8).


**Figura 6.6:** *Inicio de sesión en modo oscuro.*


**Figura 6.7:** *Intento de login sin haber introducido el correo electrónico.*


**Figura 6.8:** *Vista de login con datos de ejemplo.*

Una vez habiéndose autenticado, se carga el menú principal de la aplicación (figura 6.10). En él, cómo se ha explicado anteriormente, se pueden ver una barra superior `AppBar` con las opciones de **cerrar sesión** y los dos puntos, que permiten **cambiar de tema**.

La parte inferior de la pantalla contiene un `BottomNavigationBar`, en el que se puede cambiar de pestañas. Esto se gestiona desde `PaginaPrincipalPage`, que define varios conceptos clave para que se solo se hagan actualizaciones de eventos cuando sean necesarias.

Primero, se usa la clase abstracta `GlobalKey`, con el estado actual de la página, con el objetivo de que cuando haya que actualizar los eventos, se reciba la señal de hacerlo desde cualquier punto de la aplicación. De la misma manera, también se gestiona el número de página actual con un `PageController` que permite tanto construir los iconos de cada página individual con el color correspondiente del tema como realizar la navegación al pulsar en los iconos o deslizar y almacenar el estado de la misma.

Además, también se define todo el estilo de la barra superior, usando una función que devuelve un `CustomPainter` personalizado usando `quadraticBezierTo` que matemáticamente hablando, utiliza una *Curva de Bézier cuadrática* que interpola suavemente dos puntos $P_0$ y $P_2$ dado un punto intermedio $P_1$, que se define con la siguiente fórmula:

$$B(t) = (1-t)^2 P_0 + 2(1-t)tP_1 + t^2 P_2, \quad t \in [0,1]$$

De manera gráfica, una curva de Bézier cúbica tendría la forma de la figura 6.9, que es lo que se busca, pero que se aplica de manera más suavizada en AgroFind, utilizando el código ya existente del repositorio de Luis García:
https://github.com/alejandrolgarcia/designs-flutter
Se utiliza esta técnica para dar una imagen más profesional y estilizada a la aplicación, haciendo que el gradiente entre los dos colores principales sea mucho más suave y preciso.

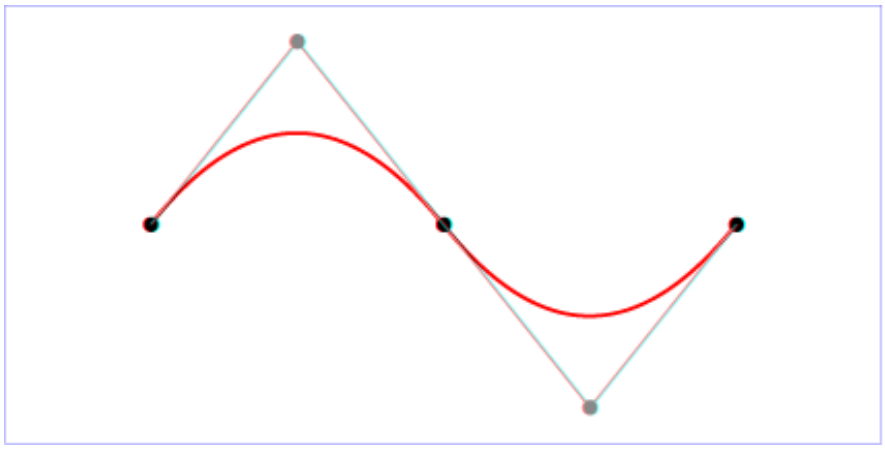


**Figura 6.9:** *Ejemplo gráfico de curva de Bézier. Fuente:* https://exploringswift.com/blog/Drawing-Smooth-Cubic-Bezier-Curve-through-prescribed-points-using-Swift.

Respecto al círculo de carga, se ha animado el logo, usando `AnimationController` (figura 6.10). Esta función está definida en el archivo de utilidades, del que se irán explicando varias otras funciones reutilizadas a lo largo del código para evitar duplicidades lo máximo posible.

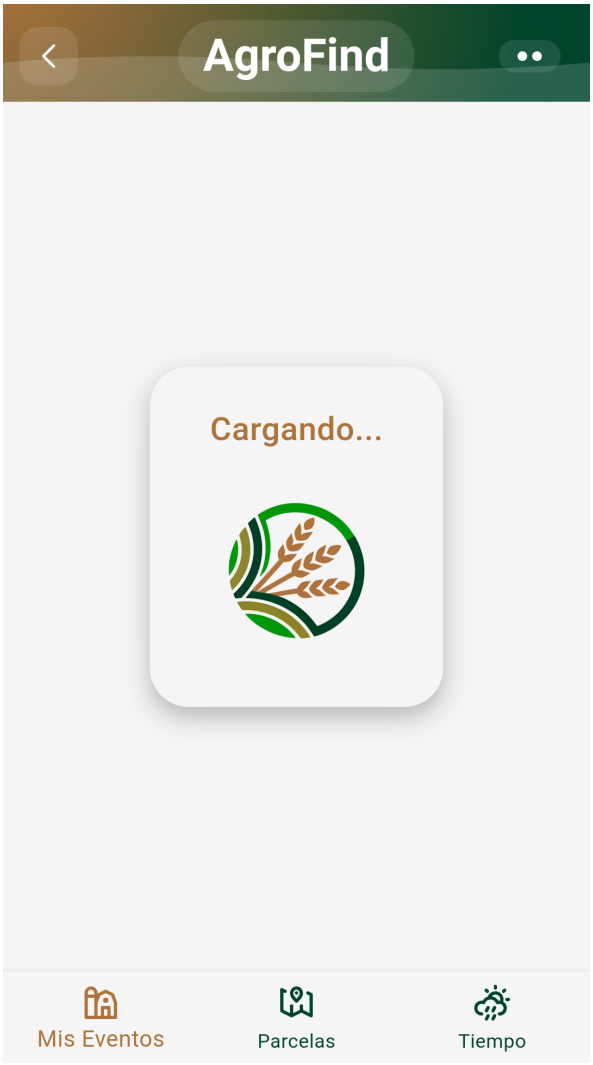



**Figura 6.10:** *Cargando eventos en la pestaña principal.*

### Home / Lista de eventos

En la pestaña de **Mis Eventos**, se carga la página **PaginaEventosPage**, que es responsable de cargar los eventos en la base de datos en caso de que no estén ya, llamando a la API y obteniéndolos. Además, muestra los eventos en bloques distinguidos por cada parcela, enseñando los datos más importantes de estos con físicas de scroll, mientras que si no hay eventos, aparece un texto indicativo de ello, cómo en la figura 6.11. Por cada evento aparece el tipo del mismo y su actividad, que se explicarán más adelante.

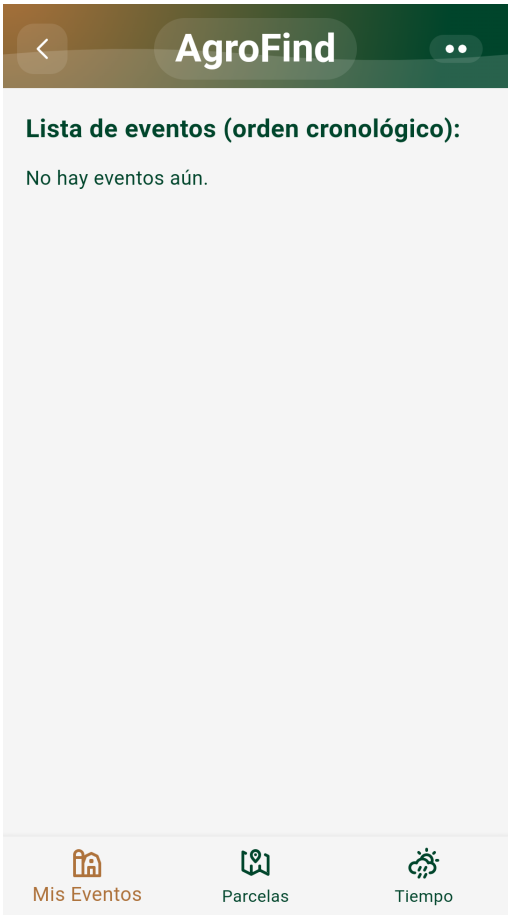



**Figura 6.11:** *PaginaEventosPage, sin ningún evento.*


**Figura 6.12:** *PaginaEventosPage, con eventos realizados.*

Los eventos se cargan llamando a AgroFindAPI, a la ruta `/obtener-eventos` con el idUsuario como parámetro. Esta invoca a la Lambda **ObtenerParcelasDynamo** (véase 6.2.2) que llama al índice global o GSI de la tabla, que devuelve la información necesaria para cargar los eventos y se hace con un método GET (extracto de código 6.6). La arquitectura de la misma ha sido analizada previamente en la sección 5.2.4.

```
Future<List<Evento>> cargarEventosDesdeDynamo() async {
    try {
      final respuesta = await http.get(
        Uri.parse('${dotenv.env['API_URL_OBTENER_EVENTOS']}?idUsuario=${
    _usuarioActual!.uid}'),
        headers: {'x-api-key': dotenv.env['API_AGROFIND_KEY']!},
      );
      if (respuesta.statusCode == 200) {
        List<dynamic> info = json.decode(respuesta.body);
        for (var item in info) {
          try {
            Evento unEvento = Evento.fromJson(item);
            unEvento.parcela = await _servicioParcela.
               obtenerParcelaPorId(unEvento.idParcela);
            _listaEventos.add(unEvento);
            await _servicioEvento.guardarEvento(unEvento);
/* resto de código */
```

**Extracto de código 6.6:** *Función cargarEventosDesdeDynamo.*

Una vez visto como se cargan los eventos, se explicará más adelante que utilidad tienen, ya que ahora hay que analizar los dos mapas que hay en AgroFind.

### Mapa de parcelas propias

Cuando el agricultor accede a la pestaña de **Parcelas**, lo primero que se carga es la **confirmación del uso de la ubicación** (figura 6.13) por parte de la aplicación, ya que es necesario que se acepte. Es obligatorio para que la aplicación funcione en su totalidad, y para evitar posibles fraudes, si no se acepta este aviso, la app se cierra de manera natural y controlada. Al volver a entrar al sistema vuelve a salir el mismo aviso.

Si se acepta, se carga el mapa de la figura 6.14, que tiene un flujo de información muy similar al de obtener los eventos, salvo que en este caso es necesario recoger todas las parcelas, ya que hay que mostrar en los mapas las parcelas que no se pueden reclamar para que estén bloqueadas. Esto lleva a que no sea necesario un índice global, sino que con la función `scan` que sería equivalente a obtener toda la tabla, es suficiente.


**Figura 6.13:** *Confirmación de acceso a la ubicación.*

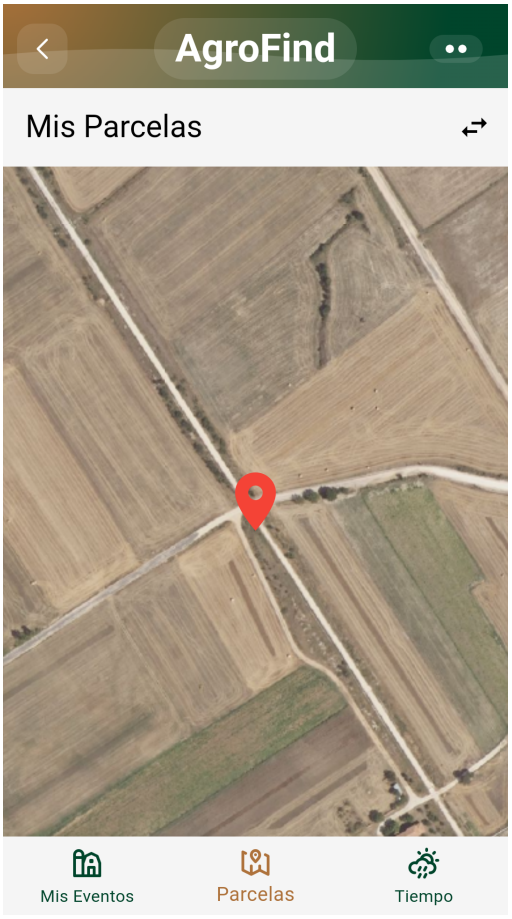



**Figura 6.14:** *Mapa de mis parcelas sin ninguna parcela propia.*

Hasta que el agricultor no tenga parcelas añadidas, el mapa de mis parcelas estará vacío, por tanto, es necesario cambiar al mapa de SIGPAC. Para ello, se pulsa el botón superior derecho de la pestaña y aparece un `AlertDialog` (figura 6.15) con las opciones disponibles para elegir el mapa. Si se pulsa uno que ya estaba seleccionado no se vuelve a cargar, sino que simplemente se cancela el diálogo. En cambio, si se pulsa el mapa contrario, se carga este en el lugar del anterior. Aún así, se almacena la información cargada en el mapa de SIGPAC en caché (véase sección del mapa SIGPAC: 6.1.4).

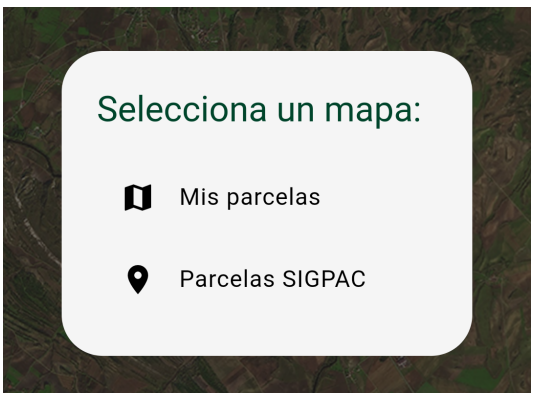



**Figura 6.15:** *Diálogo de cambiar el mapa.*

### Mapa SIGPAC

Esta es una de las secciones que más complejas son del trabajo, debido a que el acceso a los datos SIGPAC ha sido muy enrevesado, especialmente por las limitaciones de su API.

En este proyecto las palabras parcela y recinto se utilizan con el mismo significado, pero *para el FEGA, las parcelas pueden contener varios recintos*, y por tanto, en AgroFind, cuando se hace referencia a una parcela, siempre se está refiriendo al recinto concreto de una parcela, porque en el caso contrario, no se podría hacer esa división a posteriori.

Primeramente, se ha tenido que generar una capa personalizada para poder cargar las imágenes que llegan desde el servicio WMS de SIGPAC (véase 2.1), pero para ello ha habido que hacer un análisis extenso sobre la capacidad del servidor, que en los servicios WMS se suele denominar como `getCapabilities` [82]. Se ha obtenido la conclusión de que aunque aparezcan muchas proyecciones geográficas disponibles para usar, éstas varían dependiendo de cada servicio dentro del WMS, entre los que se encuentra **AU.Sigpac:recinto**, que es el que se empleará, pero hay otros tantos más como *elementos del paisaje, puntos del paisaje o línea declarativa* que se podrían usar en el futuro para dar más detalle al mapa.

La capa personalizada es `SigpacCuadradosLayer`, y está definida en una clase aparte que, por resumir su implementación, es un componente que gestiona la visualización del SIGPAC utilizando `TileLayer` (una *tile* es una baldosa, en este caso, una parte concreta del mapa). Implementa un sistema de caché mediante `CacheManager` (véase 6.7) para almacenar los archivos descargados durante un tiempo definido [83] y limita el número máximo de objetos en caché a 200. Además, utiliza una cola `Queue` para gestionar las solicitudes de descarga de `tiles`, evitando saturar el sistema mediante un procesamiento secuencial con un pequeño retraso entre cada solicitud.

```
final _gestionCache = CacheManager(
  Config(
    'sigpac_layer_cache',
    stalePeriod: const Duration(days: 7),
    maxNrOfCacheObjects: 200,
  ),
);
```

**Extracto de código 6.7:** *CacheManager de la clase SigpacCuadradosLayer.*

[82] En el caso del SIGPAC véase [101].

[83] Este periodo se ha definido como 7 días, para que en caso de algún cambio de recintos por parte del FEGA, no se quede para siempre la misma versión.

La clase también incluye un proveedor de tiles o `TileProvider` personalizado llamado `SigpacCuadradosTileProvider` que se encarga de generar las URLs de los tiles y gestionar su descarga cómo se ve en el extracto de código 6.8, asegurando que las imágenes se carguen desde la red o desde la caché según sea necesario. Esto permite una integración eficiente y controlada de la capa de los recintos en la aplicación, y hace que se cargue lo más rápido posible.

Otra función importante, `_procesarCola`, es recursiva, y va procesando todos los mensajes que se han ido introduciendo a la `Queue` desde el provider, introduciendo un delay para evitar sobrecarga de la API de SIGPAC.

```
return TileLayer(
  tileProvider: SigpacCuadradosTileProvider(
    gestionCache: _gestionCache, colaPeticiones: _colaPeticiones,
    procesarCola: _procesarCola,
  ),
  tileDisplay: TileDisplay.fadeIn(),
  wmsOptions: WMSTileLayerOptions(
  baseUrl: "https://wms.mapa.gob.es/sigpac/ows?",
  layers: ['AU.Sigpac:recinto'],
  version: '1.3.0', format: 'image/png',
  transparent: true, crs: const Epsg3857(),
  styles: [''], otherParameters: {'wmtver': '1.3.0',
    'scaleMethod': 'accurate', 'TILED': 'true'},
  ),
);
```

**Extracto de código 6.8:** *TileLayer que devuelve SigpacCuadradosLayer.*

Todo ello se utiliza en la clase `LocalizacionPage`, donde para cargar el mapa SIGPAC hay 4 componentes.

1. Una `TileLayer` con los datos del IGN/CNIG, que es una capa de la península ibérica donde dependiendo de que altura de zoom esté, se cambia de tipo de visualización. A esto se le llama *ortofoto* (véase 2.2) y es la base para que el mapa tenga información de satélite reconocible para los agricultores. Se obtiene a partir del servicio WMS que ofrece el IGN [102] utilizando `WMSLayerTileOptions` para personalizar las peticiones.

2. La `SigpacCuadradosLayer` ya explicada en esta sección, en el que se verán los límites entre parcelas en un color rosa fosforito, para hacer de contraste en cualquier tipo de terreno del mapa, como se puede ver en la figura 6.16.

3. Una `PolygonLayer` creada a partir de las parcelas que están guardadas, es decir, todas las parcelas de todos los agricultores, no solo las que sean del usuario actual. Estas últimas se dibujarán en el color principal del tema (figura 6.17), mientras que las ya reclamadas por otros se dibujarán en el color secundario y no se podrá hacer ninguna acción al tocar en ellas.

4. `Marker` en la ubicación actual, desde el que se calcula la distancia a las parcelas.


**Figura 6.16:** *Mapa SIGPAC en el que no hay ninguna parcela reclamada por ningún agricultor.*

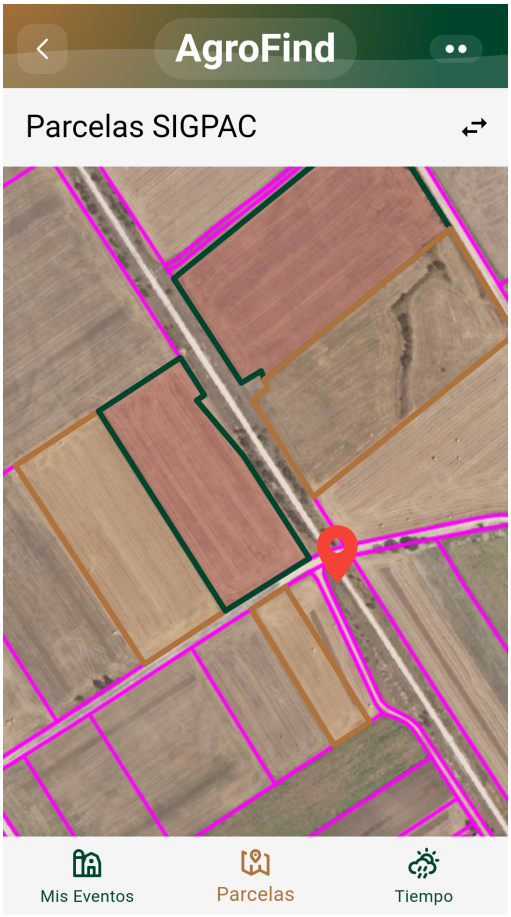



**Figura 6.17:** *Mapa SIGPAC con múltiples parcelas reclamadas.*

Cuando se pulsa o se mantiene pulsado en cualquier punto del mapa (ambas opciones para mejor usabilidad), aparece un diálogo como el de la figura 6.18, donde si el usuario quiere simplemente obtener información de esa parcela pero no reclamarla puede hacerlo. Por otra parte, si va a reclamar la parcela, aparece otro diálogo de confirmación (figura 6.19), para evitar errores a la hora de añadir parcelas, debido a que el agricultor no puede quitar sus parcelas en ningún momento.

En este punto habría que comprobar si realmente esa parcela pertenece a ese agricultor (o la ha alquilado), y la manera más veraz de hacerlo sería a través del uso del DNI electrónico como método de identificación segura[84].

Para poder tratar los datos personales extraídos del certificado digital (como el nombre o el NIF), es imprescindible cumplir con la normativa vigente sobre de protección de datos[85]. Debe garantizarse la legalidad del tratamiento de la información y la verificación de los certificados electrónicos conforme a la ley[86]. Por tanto, se deja esta funcionalidad pendiente como una mejora a futuro necesaria si la aplicación se llegase a poner en producción, ya que es lo que le daría un componente de más veracidad a los datos, al cotejarlos con la base de datos interna tanto del Departamento General de la Policía como los del MAPA.

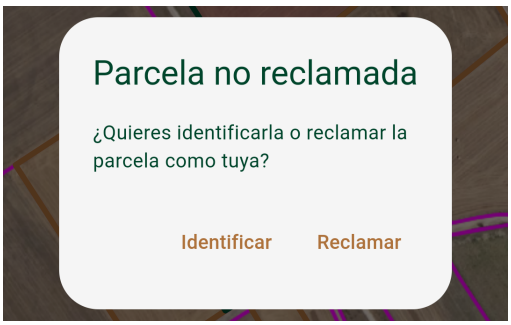



**Figura 6.18:** *Diálogo de parcela no reclamada.*


**Figura 6.19:** *Diálogo para asegurar el reclamo de una parcela.*

Además, tanto si se pulsa en una parcela del agricultor registrado como en cualquiera de las otras que estén ya reclamadas, aparece otro diálogo impidiendo su reclamación (figuras 6.20 y 6.21).

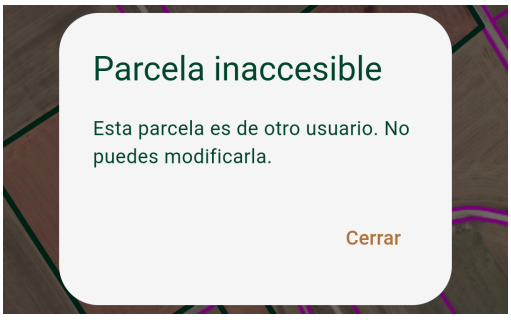



**Figura 6.20:** *Diálogo de parcela de otro usuario.*

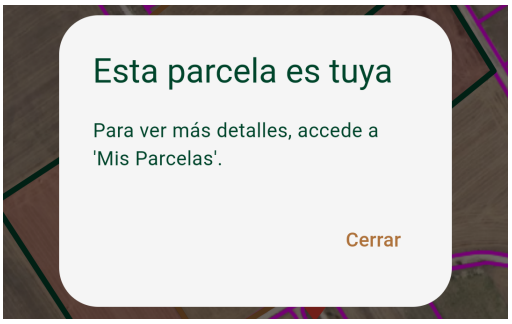



**Figura 6.21:** *Diálogo de parcela ya reclamada por el usuario.*

[84]Parlamento Europeo y Consejo de la Unión Europea. Reglamento (UE) Nº 910/2014 sobre identificación electrónica y los servicios de confianza (eIDAS) [103].

[85]Parlamento Europeo y Consejo de la Unión Europea. Reglamento General de Protección de Datos (RGPD), 2016 [104]. Véase también la Ley Orgánica 3/2018 de Protección de Datos y garantía de los derechos digitales (LOPDGDD) [105].

[86]Gobierno de España. Ley 6/2020 reguladora de los servicios electrónicos de confianza [107].

Si el agricultor decide identificar una parcela, obtendrá un informe breve sobre la misma, donde aparece toda la información relevante, y lo más crucial, el uso SIGPAC. Como se ha comentado previamente, de esto depende si la parcela es cultivable o no, y por tanto, desde ahí se puede consultar esto mismo. En la figura 6.22 se ve una parcela común y corriente, que es arable y por tanto susceptible a poder usarse, al igual que otros usos SIGPAC cómo *pasto arbustivo, huerta, frutales, pastizal...*

En la figura 6.23 aparece una parcela de *zona urbana*. Esto quiere decir que no es cultivable directamente, debido a que según el catastro, no existe un campo declarado en ese terreno. Este tipo de parcelas se pueden reclamar, pero no se les pueden añadir eventos, así como en otros casos (*elemento del paisaje, parcela improductiva, zona vial, terreno urbanizado...*).


**Figura 6.22:** *Información de una parcela no reclamada.*

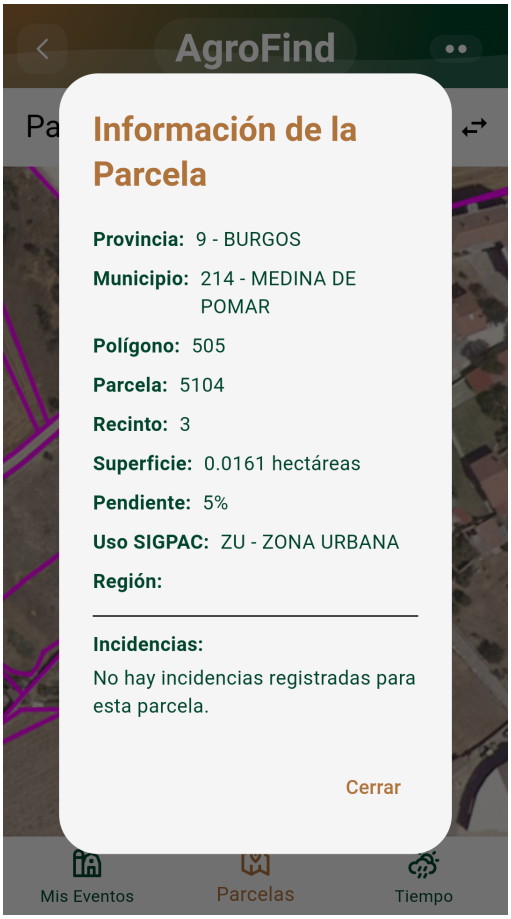



**Figura 6.23:** *Información de una parcela con terreno inválido no reclamada.*

### Detalles de una parcela

Para poder acceder a los detalles de las parcelas, hay que acceder a través del mapa de **Mis Parcelas**. Habiendo tocado en una de ellas, aparece un diálogo con la información de la misma de manera resumida cómo se ve en la figura 6.24, donde se encuentra el nombre, el id de la parcela, su uso SIGPAC y el área, pero esto no son los detalles, sino que para llegar a esa interfaz hay que pulsar en el botón de **Ver Detalles**.

Para que el programa detecte si donde el usuario ha hecho *tap* es una de las parcelas, se utiliza el algoritmo de **Ray-Casting**, que se suele emplear en videojuegos, pero en este caso tiene sentido para hacer las comparaciones de manera más óptima. El algoritmo se basa en lanzar un rayo horizontal desde el punto a comprobar hacia el infinito y contar cuántas intersecciones tiene con los lados del polígono (véase [108]). Matemáticamente se define de la siguiente manera:
Sea $P = (x, y)$ y un polígono con vértices $V_i = (x_i, y_i)$, $i = 0, \ldots, n-1$, con $V_n = V_0$.

Se define el número de intersecciones del rayo horizontal desde $P$ con los lados del polígono cómo:

$$N = \sum_{i=0}^{n-1} \left[ (y_i < y \leq y_{i+1} \text{ o } y_{i+1} < y \leq y_i) \quad \wedge \quad x < x_i + \frac{(y - y_i)(x_{i+1} - x_i)}{y_{i+1} - y_i} \right]$$

El punto $P$ está dentro del polígono si $N$ mód $2 = 1$, y fuera de él si $N$ mód $2 = 0$.


**Figura 6.24:** *Diálogo al pulsar en una parcela.*

La página de detalles de parcela (`DetallesParcelaPage`) se compone de varios componentes en la interfaz visual que se puede ver en la figura 6.25.

1. El `Title` es el id de la parcela y debajo aparece una sección con el nombre de la misma, el uso SIGPAC correspondiente, el área, la latitud y la longitud, todo en una `Column` dentro de un `Scaffold`.

2. Hay dos botones (tipo `ElevatedButton`) que realizan acciones relevantes, el primero lleva a *Google Maps* [87] a la ubicación de la parcela correspondiente, para que el agricultor pueda llegar a la misma en caso de haberla adquirido recientemente o para poder compartir la ubicación de manera más sencilla.
3. Respecto al segundo, este permite cambiar el nombre a la parcela tanto en la base de datos local como en DynamoDB, para que el agricultor reconozca los campos por el nombre que más le convenga, que para este ejemplo será *Terreno Sur*. El flujo de información para ello es el que ha presentado anteriormente en la figura 5.17.
4. La parte inferior de la pantalla contiene una `ListView` con todos los eventos, donde para cada uno se construye una `Card` personalizada que tiene un `Icon`, el tipo de evento que ha sido, la fecha exacta y la información extra sobre el evento, llamado actividad. Todo esto se construye llamando al método `_construirListaEventos`.
5. El elemento final de esta interfaz es el botón de **Añadir Evento**, que se trata de un `FloatingActionButton` [88]. Cuando se pulsa, se hacen dos comprobaciones principales en la función `_mostrarFormularioEvento`, entre las que está la distancia y el tipo de uso.

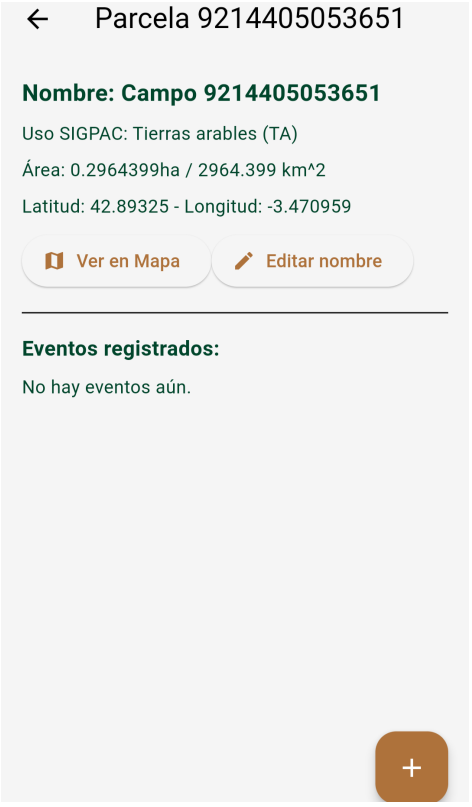



**Figura 6.25:** *Detalles de parcela sin ningún evento asociado.*


**Figura 6.26:** *Cambio de nombre en detalles evento parcela.*


**Figura 6.27:** *Cambio de nombre de una parcela en mis parcelas.*

[87] Sería similar a un `Intent` en Java, pero que en Flutter se define con una URL especial que abre la aplicación de Google Maps directamente haciendo la llamada a su API fuera de AgroFind.
[88] Estilización del mismo obtenida de un blog en Medium. Enlace: https://medium.com/@wartelski/flutter-3-floating-action-buttons-949be319df82

### Añadir evento

Para poder añadir un evento, el agricultor no puede estar a mas de 2 km de la parcela, para evitar posibles fraudes con los eventos y además, la parcela tiene que tener un uso SIGPAC que permita labrarla y realizar actividad en ella (véase 6.1.4). En cambio, si se cumplen las condiciones, aparece un `ActionDialog` con todas las acciones necesarias para añadir un evento nuevo. Estas son el tipo (figura 6.28) , que puede ser **siembra**, **arado**, **riego** (si es necesario en ese tipo de producto) y el más importante, el de **cosecha**, que en el código es una lista desplegable (DropdownButton) como se puede ver en la figura 6.29.

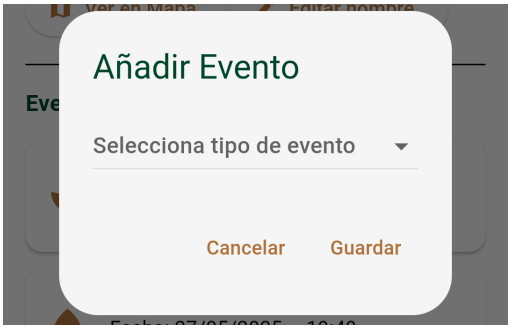



**Figura 6.28:** *Primer paso de añadir evento.*

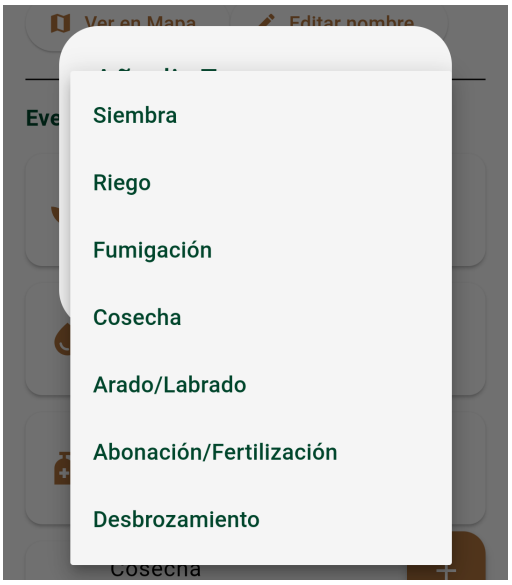



**Figura 6.29:** *Segundo paso de añadir evento.*

Además se elige una actividad asociada (figura 6.30), para darle más precisión al evento, que es una segunda lista desplegable que depende del tipo de evento seleccionado. Por ejemplo, para **siembra**, las actividades incluyen *Alfalfa, Arroz, Avena...* en total unos 25 productos a elegir, con la opción especial de *no especificado* por si es alguno de los que no se ha tenido en cuenta.

Una vez elegidas las opciones como en la figura 6.31, se pulsa en guardar y automáticamente el evento se guarda en la blockchain, en local y en la base de datos remota. Como este proceso tarda un tiempo de entre 2 y 5 segundos para cualquier evento y hasta 10 segundos para los eventos de cosecha porque hay que generar un PDF con toda la información, aparece una pantalla de carga similar a lo que se ha visto anteriormente.


**Figura 6.30:** *Tercer paso de añadir evento.*

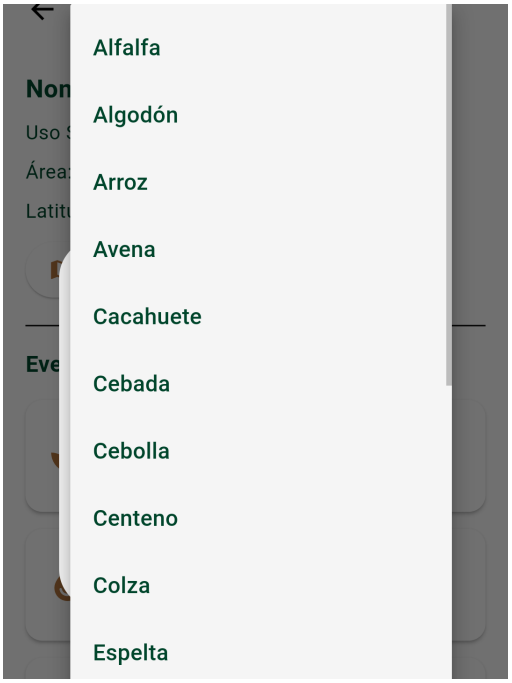



**Figura 6.31:** *Cuarto paso de añadir evento.*

Para ver todo el flujo de información de añadir un nuevo evento a la cadena de bloques, hay que tener en cuenta que primero se envía la transacción y después se tiene que añadir a la base de datos local y a la tabla de DynamoDB, cuyo flujo de información se ha visto anteriormente en la sección de arquitectura (véase 5.2.4).

### Certificado QR

Cuando se realiza una cosecha en una parcela, el sistema comprueba si se ha hecho una siembra previamente, y en caso afirmativo, usando el tipo de producto sembrado, se guarda en la blockchain el evento, pero además, en AWS se crea un **un informe PDF personalizado para esa cosecha** y localmente se crea un QR que lleva a la ruta donde se almacena este documento. Esto se muestra en la interfaz con la figura 6.32, utilizando la clase **`QRCertificadoPage`**, en la que se encuentra la imagen del código QR, el enlace debajo para poderlo copiar y pegar directamente en el navegador en caso necesario y un botón donde se puede descargar el código, siempre pidiendo permiso al usuario (figura 6.33), para permitir guardarlo en el dispositivo. Si se dan los permisos, aparece un `SnackBar` que avisa de que se ha almacenado en la galería (figura 6.34).


**Figura 6.32:** *Interfaz en la que se muestra el certificado QR.*


**Figura 6.33:** *Permisos para guardar el código QR en el dispositivo.*

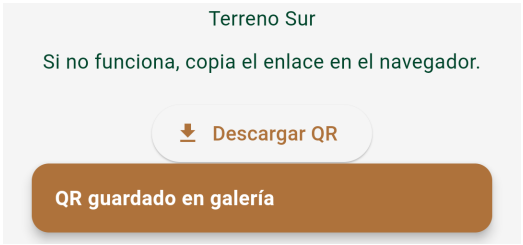



**Figura 6.34:** *Snackbar con aviso de QR guardado en galería.*

La estructura del PDF, su visualización y el flujo de información del mismo se explicará en el apartado 6.2.2 y en la figura 6.43, al igual que el flujo de añadir el evento, puesto que tiene una complejidad mayor y depende tanto de AWS como de la blockchain de Polygon. De todos modos, se adjunta un enlace para acceder a un certificado de prueba:

agrofind-certificados.s3.eu-south-2.amazonaws.com/certificados/9214405053801.pdf

### Tiempo

La página del tiempo muestra información meteorológica actual para la ubicación del usuario y las parcelas asociadas a su cuenta. El método `_cargarTiempo` es responsable de obtener el clima actual y utiliza la API de **WeatherAPI** para realizar una solicitud HTTP, recuperando datos como la temperatura, velocidad del viento y un ícono representativo de las condiciones climáticas. Estos datos se procesan y actualizan en la interfaz, al igual que los datos individualmente para parcela usando un `ListView` personalizado, que siguen un proceso similar. A continuación se pueden ver dos ejemplos, uno con la ubicación actual (figura 6.35) [89] y otro con una ubicación en concreto (figura 6.36), que puede ser de cualquier punto de la península.


**Figura 6.35:** *Tiempo en ubicación actual.*


**Figura 6.36:** *Tiempo habiendo buscado una ubicación personalizada.*

[89] Nótese que el tiempo no aparece en Moneo sino en Bustillo de Villarcayo, esto se debe a que WeatherAPI agrupa por proximidad ciertas poblaciones y no da el nombre de Moneo directamente.

## 6.2. Backend

Al ser una aplicación sin servidor, el backend no es una máquina virtual donde hay una base de datos o una herramienta expuesta en ciertos puertos, sino que es una serie de servicios externos entre los que se encuentran Firebase, AWS, Polygon, SIGPAC, IGN/CNIG, WeatherAPI...
En este capítulo se explicará cada uno de ellos en detalle, y los que no tengan una sección propia se analizarán donde corresponda, es decir, la información obtenida tanto de SIGPAC cómo IGN/CNIG ya se ha detallado anteriormente en el capítulo de interfaz de usuario (6.1.4), pero se matizará su uso en este.

Como ejemplo, **SIGPAC** es el sistema que se ha utilizado para poder visualizar las parcelas en el mapa de la aplicación, pero de este se han utilizado varios servicios, entre los que destacan WMS y SIGPAC Hubcloud. El primero sirve para la descarga de las imágenes (véase 6.16, que vienen en un formato de `256x256 píxeles`, y el segundo se encarga de obtener detalles específicos de una parcela en concreto.
Cuando se va a añadir una nueva parcela a un agricultor, este servicio devuelve información relevante sobre la misma, que es clave, porque de ahí se obtiene el uso SIGPAC de la parcela. Estos dos son información pública y utilizable bajo la licencia *Creative Commons 4.0*, como se ha analizado en la sección correspondiente.

También se ha llevado a cabo la pestaña del tiempo utilizando **WeatherAPI** como base, ya que tiene una capa gratuita extensa de hasta un millón de peticiones mensuales. Esto querría decir que hasta que la aplicación no supere este número de peticiones no es necesario hacer ningún pago. Haciendo una estimación de que un usuario consulta 5 veces el tiempo al día, y que tiene entre 5 y 20 parcelas, daría espacio desde 10.000 e incluso hasta 40.000 usuarios activos al sistema. Es muy fiable y se puede utilizar para más cosas en el futuro.

Finalmente, la API que ofrece el **IGN/CNIG**, es un WMS [90] que está optimizado para interactuar con librerías de mapeo como lo es *flutter_map* y por tanto permite que los mapas carguen a una velocidad muy rápida y sin requerir de una conexión a internet estable, ya que se almacenan en caché. Es un servicio abierto, y por tanto, si AgroFind se quiere llegar a emplear en producción esto sería otra de las fortalezas de la misma.

[90] Véase el *getCapabilities* del servicio PNOA. Enlace: https://www.ign.es/wms/pnoa-historico?request=GetCapabilities&service=WMS

### 6.2.1. Firebase

Respecto a la implementación del sistema de autenticación mediante Firebase, se ha empleado la consola [91] web. Se ha hecho un proyecto nuevo llamado AgroFind, al que se le ha habilitado el paquete de **Authentication**. Cómo se ve en la figura 6.37, en la pestaña de **método de acceso** se ha definido el acceso a la app por correo electrónico, y en la pestaña de **configuración**, que la aplicación es multiplataforma, de esta manera se puede usar el mismo proyecto para todas las plataformas. En **usuarios** se pueden observar los usuarios registrados, y en **plantillas** se pueden definir textos en formato HTML para cuando haya que enviar un correo electrónico al agricultor que se le haya olvidado la contraseña (figura 6.38).

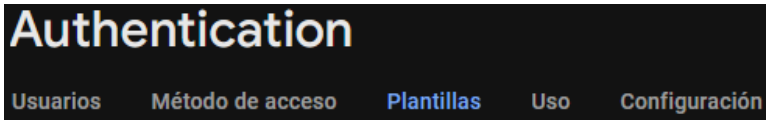



**Figura 6.37:** *Interfaz de Firebase para configurar la autenticación.*

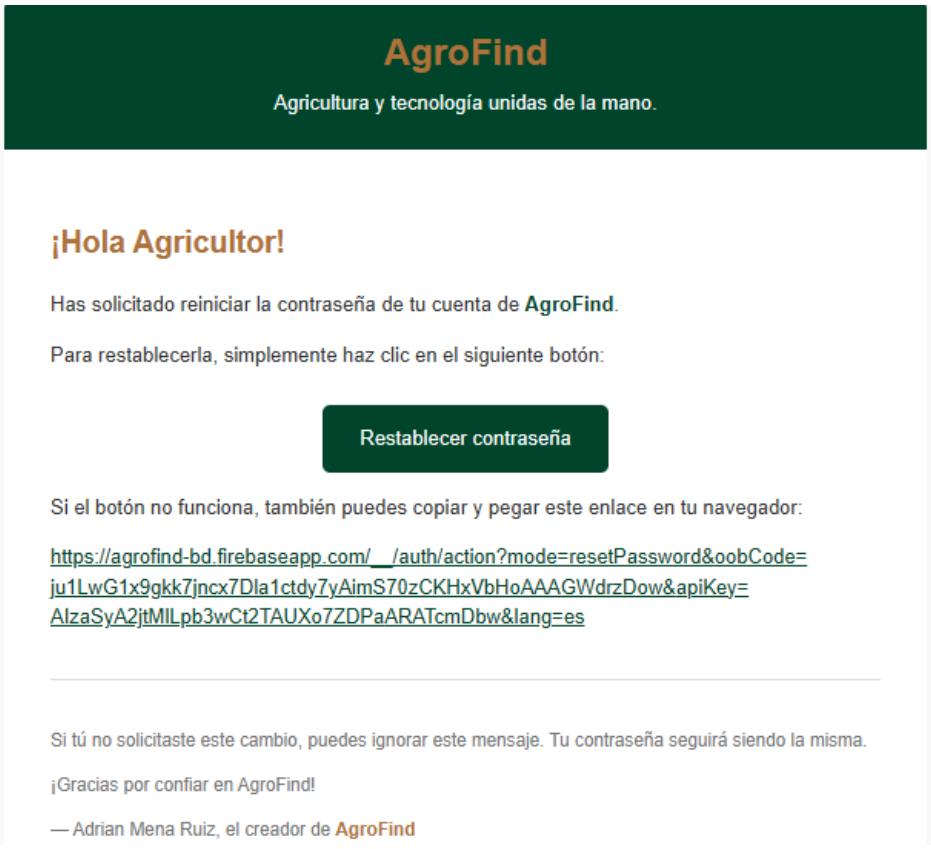



**Figura 6.38:** *Correo electrónico de recuperación de contraseña.*

[91] Enlace a la consola: https://console.firebase.google.com/

Una vez configurado el proyecto, hay que trasladar esa configuración a la aplicación, y para ello hay que descargar (desde la configuración general del proyecto) el archivo `google-services.json` (en el caso de iOS sería otro archivo similar) y colocarlo en la ruta `app/android`. Este archivo contiene todas las claves e identificadores necesarios para hacer funcionar el servicio de Firebase en Android. En caso de la app de escritorio (Windows o Linux) y en la opción web no es necesario, puesto que con el paso final se importa automáticamente. Esto consiste en utilizar la CLI previamente instalada para utilizar:

```
firebase login
```

Una vez hecho el inicio de sesión con la cuenta de Google correspondiente a la asociada al proyecto, la configuración está finalizada. Ahora para dar funcionalidad a la configuración, hay que codificar las instancias de Firebase en el código Dart. Esto se hace inicialmente en `AgroFindApp` (6.9), donde se inicializa en base a la plataforma actual:

```dart
import 'package:agrofind/firebase_options.dart';
import 'package:firebase_core/firebase_core.dart';
/* imports varios */
void main() async {
  await dotenv.load();
  WidgetsFlutterBinding.ensureInitialized();
  // inicializar instancia de Firebase
  await Firebase.initializeApp(options:
                DefaultFirebaseOptions.currentPlatform);
  runApp(AgroFindApp());
}
```

**Extracto de código 6.9:** *Uso de Firebase en la clase AgroFindApp.*

Se usa en otros puntos del código, en cualquiera de las clases que requiera usar el usuario, donde se carga en una variable que puede ser nula, por si hay algún fallo con el sistema de inicio de sesión, que el sistema no *crashee* por ello. Además, cuando se hace un registro de un nuevo usuario, se gestiona usando `createUserWithEmailAndPassword` de la instancia de Firebase, que realiza toda la lógica de crear la nueva cuenta. De la misma manera, se emplea `signInWithEmailAndPassword` para realizar el inicio de sesión. Todos estos ejemplos se presentan en el siguiente fragmento de código, que es una colección de líneas de código de distintas clases (6.10):

```dart
/* ejemplo 1 */
User? _usuarioActual = FirebaseAuth.instance.currentUser;
/* ejemplo 2 */
_usuarioActual?.uid // para obtener el identificador si existe
/* ejemplo 3 */
UserCredential? credencialUsuario =
    await FirebaseAuth.instance.createUserWithEmailAndPassword(
          email: controladorMail.text, password: controladorPass.text,
        );
/* ejemplo 4 */
    await FirebaseAuth.instance.signInWithEmailAndPassword(
        email: controladorMail.text, password: controladorPass.text,
      );
```

**Extracto de código 6.10:** *Uso de Firebase a lo largo del código.*

### 6.2.2. AWS

Como se ha podido ir viendo a lo largo de la memoria, este proyecto utiliza AWS como método principal de backend, y en esta sección se explicará para que se ha usado cada servicio en profundidad, pero sin explicar cada trozo de código uno a uno.

Es importante destacar que toda la infraestructura realizada en AWS para este proyecto está ubicada en la región de España, conocida como eu-south-2 [92], que se encuentra en varios centros de datos de Aragón. Esto se hace para que si en un futuro se llegase a implementar en producción, todo lo relacionado con la legalidad como la RGPD o la LOPD se puedan cumplimentar más fácilmente.

#### API Gateway

La API se ha implementado utilizando la opción **REST**, configurada manualmente mediante la consola de AWS. Para cada recurso se ha definido una ruta específica, asignando el método HTTP correspondiente (GET, POST, etc.) según la funcionalidad requerida. La integración de estos recursos con las funciones *Lambda* se ha hecho mediante el modo *Lambda Proxy Integration*, permitiendo que la totalidad de la petición HTTP (cabeceras, cuerpo y parámetros de consulta) se transfiera de forma directa a la función Lambda sin transformación previa por parte de API Gateway.

Cómo se ha explicado previamente, se usa una clave API, que se gestiona a través de un *Usage Plan* o pland e uso donde se han definido límites específicos de uso *Rate* y *Burst* configurados directamente desde la consola (figura 6.39).

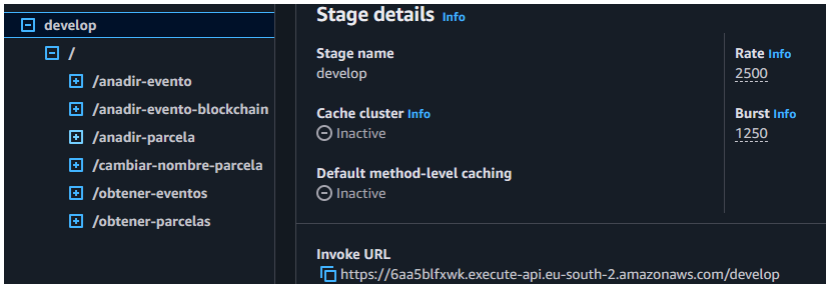



**Figura 6.39:** *Pestaña de despliegue de fases de AgroFindAPI.*

Para garantizar la compatibilidad con web y escritorio, se ha activado el soporte a *CORS* (*Cross-Origin Resource Sharing*). Además de configurar manualmente los encabezados permitidos, orígenes autorizados y métodos disponibles, se ha definido explícitamente el método **OPTIONS** en cada uno de estos recursos. Este método es gestionado directamente por API Gateway y devuelve las cabeceras necesarias para permitir el intercambio de recursos entre distintos orígenes.

El despliegue y gestión de la API se ha llevado a cabo manualmente mediante la consola de AWS, creando una única etapa de ejecución conocida como *stage* correspondiente al entorno actual del proyecto.

[92] La región Española se lanzó en 2022, con el fin de tener una alternativa más veloz y con cumplimiento legal nativo. Fuente: https://aws.amazon.com/es/about-aws/global-infrastructure/regions_az/

### Lambdas

Respecto a las lambdas, han sido el centro del backend en este proyecto ya que, gracias a su naturaleza *serverless* hace que no haya que montar un servidor propio. En el anterior apartado se han mencionado varias de ellas, pero hay otras muy relevantes que se llaman como subproducto de otra llamada. Hay que mencionar también que la mayoría de ellas están en **Python 3.11**, salvo una que está en **Python 3.9** y las dos que interactuan con Polygon en **Node.js 20**, debido a que las librerías están mejor documentadas.

Hablando sobre las librerías, para poder utilizar cada Lambda hay que incluir las librerías externas [93] mediante lo que se conocen como **layers**, que son archivos `.zip` con las librerías exportadas y para ello hay que seguir las recomendaciones de AWS [109]. A continuación se incluye un breve script (en `Powershell`, extracto de código 6.11) que se ha utilizado para crear estos archivos layer.

```
$NombreCapa = "capa-blockchain"
$VersionNode = "nodejs20.x"
$DirectorioTrabajo = Join-Path $env:TEMP "lambda_layer"
New-Item -Path $DirectorioTrabajo -ItemType Directory | Out-Null
$DirectorioNodeJS = Join-Path $DirectorioTrabajo "nodejs"
New-Item -Path $DirectorioNodeJS -ItemType Directory | Out-Null
Set-Location $DirectorioNodeJS
npm init -y --silent
npm install ethers crypto --production --silent
Set-Location $DirectorioTrabajo
$ArchivoZIP = Join-Path (Get-Location) "$NombreCapa.zip"
Compress-Archive -Path ".\nodejs" -DestinationPath $ArchivoZIP -Force
```

**Extracto de código 6.11:** *Script para crear las layers en las Lambdas.*

- **RelayerAgroFind**. Al ser una de las Lambdas más complejas, es conveniente incluir un poco de código sobre cómo funciona la parte de la interacción con la blockchain. Primero se define el ABI [94] mínimo que permita ejecutar el smart contract, junto con la dirección de contrato que es pública. Después se obtienen la clave pública y la privada desde SSM, y se llama al smart contract (extracto de código 6.12). Es imprescindible comentar que se ha empleado la librería **ethers.js** y que se ejecuta la Lambda de *GuardarEventoBlockchain* posteriormente, para tener todo el proceso modularizado en componentes diferentes.

[93] Como gestionar librerías externas en Python. Fuente: https://docs.aws.amazon.com/lambda/latest/dg/python-layers.html.

[94] La Application Binary Interface (Interfaz Binaria de Aplicación) o ABI es el modo estándar de interactuar con contratos en el ecosistema Ethereum, tanto desde fuera de la blockchain como en interacciones contrato-contrato. Fuente: https://solidity-es.readthedocs.io/es/latest/abi-spec.html.

```
/* Codigo muy simplificado */
const CONTRATO = '0x443A1fA4bF2a8B70529dda692518889Cc45ff5D1';
const ABI = [
    "function registrarEvento(bytes32 hashEvento,
                    bytes32 hashParcela) external"
];

/* codigo intermedio */
export const handler = async (event) => {
    try {
        const clavePrivada = await obtenerClavePrivada();
        const proveedor = new ethers.JsonRpcProvider(RPC_POLYGON);
        const wallet = new ethers.Wallet(clavePrivada, proveedor);
        const ctr = new ethers.Contract(CONTRATO, ABI, wallet);
        const tx = await ctr.registrarEvento(hashE, hashP);

        const datosTx = {
          txHash: tx.hash,
          hashEvento: hashEvento,
          hashParcela: hashParcela,
          explorerLink: 'https://amoy.polygonscan.com/tx/${tx.hash}'
        };
        /* codigo intermedio */
        const command = new InvokeCommand({
            FunctionName: 'GuardarEventoBlockchain',
            InvocationType: 'RequestResponse',
            Payload: Buffer.from(JSON.stringify(payload))
        });
    /* resto de codigo */
```

**Extracto de código 6.12:** *Codigo fuente simplificado de RelayerAgroFind.*

- **GuardarEventoBlockchain**. Utiliza las librerías de **reportLab** y **PIL** [95], y este PDF se mostrará en el apartado de S3. El código se divide en varias funciones:

  1. `preparar_marca_agua`: Ajusta la opacidad del logo y lo rota para utilizarlo como marca de agua en el PDF.
  2. `generar_qr_con_url`: Genera un código QR a partir de una URL y lo devuelve como imagen en un buffer de memoria.
  3. `generar_pdf`: Crea un certificado en formato PDF usando los datos del evento, incorporando logo, marca de agua y código QR.
  4. `subir_a_s3`: Sube el PDF generado a un bucket de Amazon S3 y devuelve su URL pública.
  5. `lambda_handler`: Actúa como función principal en AWS Lambda, gestionando la entrada, generando el PDF, subiéndolo a S3 y devolviendo el resultado.

[95] Ha habido varios problemas con esta última librería, debido a las recientes actualizaciones de la misma, y para ello se ha utilizado la solución de un hilo de StackOverflow. Fuente: https://stackoverflow.com/questions/70795319/no-module-named-pil-visual-studio-code-error.

- **GuardarParcelaEnDynamo** y **GuardarEventoEnDynamo** tienen un código similar, en el que se cambia lo que se le envía en la petición en base a si hay que guardar una parcela como en el extracto 6.13, o almacenar un evento.

```
/* resto de codigo */
def convertir_coordenadas_a_decimal(lista_coords):
    lista_convertida = []
    for par in lista_coords:
        lat = Decimal(str(par[0]))
        lng = Decimal(str(par[1]))
        lista_convertida.append([lat, lng])
    return lista_convertida
/* resto de codigo */
item = {
    'idUsuario': idUsuario,  # clave de particionado
    'idParcela': idParcela,  # clave de ordenacion
    'nombre': datosParcela.get('nombre', ''),
    'informacion': informacion,
    'coordenadas': coordenadas,
    'fechahora': datetime.now().isoformat()
}
tabla.put_item(Item=item)
```

**Extracto de código 6.13:** *Guardar una nueva parcela en DynamoDB, Lambda GuardarParcelaEnDynamo.*

- **ObtenerEventosDynamo** y **ObtenerParcelasDynamo** funcionan de manera similar, utilizando la función `get` de DynamoDB, y utilizando un `serializer` o *serializador* (6.14), que pasa todas las instancias de tipo `Decimal` a `float`, para poderlas añadir sin problemas en el JSON que se devuelve a la aplicación.

```
def default_serializer(obj):
    # serializador para pasar instancias Decimal
    # a floats, para poderlas devolver
    if isinstance(obj, Decimal):
        return float(obj)
    elif isinstance(obj, list):
        return [default_serializer(e) for e in obj]
def limpiar_item(item):
    item_filtrado = {}
    for clave, valor in item.items():
        item_filtrado[clave] = valor
    return item_filtrado
def lambda_handler(event, context):
    response = tabla.scan()
    items = response.get('Items', [])
    items_limpios = []
    for item in items:
        items_limpios.append(limpiar_item(item))
    return {
        'statusCode': 200,
        'body': json.dumps(items_limpios,
              default=default_serializer)
    }
```

**Extracto de código 6.14:** *Obtener eventos o parcelas.*

- **CambiarNombreParcela**. Se utiliza la función `update_item` de la librería nativa de AWS para DynamoDB (6.15), donde dado el idUsuario y el idParcela, se hace una operación similar a un `Update` en SQL, donde se actualiza el nombre de la parcela con el nuevo nombre.

```
/* resto de codigo */
tabla.update_item(
    Key={
        'idUsuario': idUsuario,
        'idParcela': idParcela
    },
    UpdateExpression="SET nombre = :nuevoNombre",
    ExpressionAttributeValues={
        ':nuevoNombre': nuevoNombre
    }
)
/* resto de codigo */
```

**Extracto de código 6.15:** *Actualización de nombre de parcela en la Lambda CambiarNombreParcela.*

Hay que mencionar que la configuración de ejecución de todas las Lambdas es la que aparece en la figura 6.40 [96], ya que no hay que computar muchos datos y por tanto no es necesaria ni más memoria ni más tiempo de ejecución.

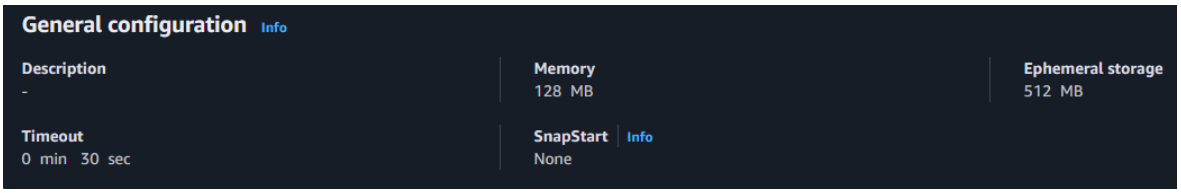



**Figura 6.40:** *Configuración de ejecución de las Lambdas.*

[96] Explicación de que es cada concepto de la configuración de las lambdas en la documentación oficial de AWS. Enlace: https://docs.aws.amazon.com/lambda/latest/dg/lambda-functions.html

### DynamoDB

Para optimizar al máximo y gastar lo mínimo posible sin eliminar funcionalidad se ha seguido el artículo de optimización de costes de la guía de developers de AWS [110]. Es importante destacar el estimado que se ha hecho en la evaluación económica, sección de costes de software (4.4), debido a que ahí se hace un presupuesto siguiendo las recomendaciones citadas.

Más concretamente, DynamoDB ofrece una flexibilidad muy importante respecto a la estructura de los objetos de sus tablas, que es perfecta para Agrofind, ya que existen objetos con estructuras de datos grandes como los *WKT*, y para ello no es adecuado fijar un tipo de objeto de antemano. Todas las tablas tienen las claves como `Strings`, y el resto de objetos pueden ser `List`, `Map`... cómo en la imagen de la figura 6.41.


**Figura 6.41:** *Ejemplo de datos de una parcela en la tabla de DynamoDB.*

Respecto a la estructura interna de esta base de datos, es muy recomendable leer el artículo publicado en la *USENIX Annual Technical Conference* [111], en el que se explica en detalle cómo funciona por dentro. Cómo se explica en ese artículo, un enrutador de solicitudes **Request Router** dirige las peticiones a los nodos de almacenamiento distribuidos en distintas zonas de disponibilidad (AZ). De esta manera, se regula el flujo de peticiones para mantener un rendimiento constante y equilibrado. Esto se puede ver más visualmente en la figura 6.42

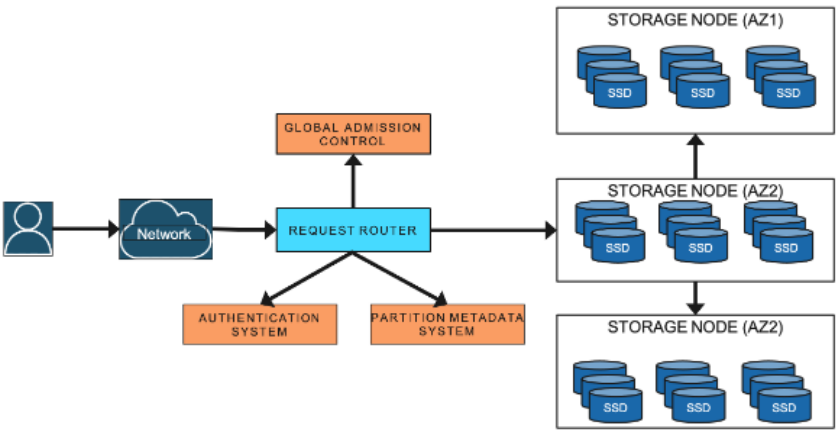



**Figura 6.42:** *Arquitectura interna de DynamoDB. Fuente: [111].*

### S3

Simple Storage Service permite almacenar de forma gratuita hasta 5 GB en su almacenamiento *Standard*, y se puede configurar cada *bucket* de múltiples maneras, pero en el caso de AgroFind interesan dos en concreto.

1. Bucket **agrofind-certificados**, que es público y por tanto cualquier persona con el enlace de uno de los elementos puede acceder a ese elemento en concreto. En esta opción se almacenan los certificados de los que se generan los QR posteriormente en local, y cada enlace es accesible como si fuera una página web alojada, solo con el detalle de que directamente se descarga el archivo PDF [97] al dispositivo. Se puede acceder al ejemplo desde: agrofind-certificados.s3.eu-south-2.amazonaws.com/certificados/9214405053801.pdf


**Figura 6.43:** *Ejemplo de estructura del PDF que se almacena en el bucket agrofind-certificados para una cosecha de colza.*

2. **agro-find-bucket**, privado y que se usa para almacenar copias de seguridad.

[97] La comprobación de la trazabilidad al detalle mediante el enlace a la blockchain se explica en la sección 6.2.3.

### Servicios auxiliares

- **Cloudwatch**. Se utiliza para monitorizar y registrar las ejecuciones de las Lambdas. Esto incluye logs de errores, métricas de rendimiento y datos de uso, lo que facilita la depuración y el mantenimiento del sistema. Concretamente, la AgroFindAPI lo integra también, desde dónde se puede monitorizar todas las peticiones que se realizan a la misma, y podría ser muy útil para ver picos de demanda de la aplicación.
- **SSM**. Secret System Manager se utiliza para almacenar de forma segura las claves privadas (API Keys, clave privada de la billetera que hace de relayer) y otros secretos, como URLs de servicios externos RPC de Polygon. Esto asegura que las claves nunca se expongan directamente en el código.
- **IAM**. Identity and Access Management se utiliza para gestionar los permisos de los servicios, donde las Lambdas tienen roles específicos que les permiten acceder a DynamoDB, S3 o SSM. Estas políticas se tienen que aplicar usando el *minimum access privilege* o el mínimo privilegio de acceso, para que en caso de algún tipo de error, no tengan opción a poder acceder a otros recursos, cómo explica Okta en su blog [112]:

  > The principle of least privilege (PoLP) is a minimum access policy that centrally manages and secures privileged credentials, and only allows users access to the least amount of required privileges.

  Estos controles de acceso tienen una estructura de JSON (véase extracto 6.16), y se pueden configurar desde la propia consola de IAM.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:UpdateItem"
            ],
            "Resource": "arn:aws:dynamodb:eu-south-2:RECURSO"
        }
    ]
}
```

**Extracto de código 6.16:** *Ejemplo de política de control de acceso en IAM.*

### 6.2.3. Blockchain

#### Smart contract

El contrato inteligente se llama **RegistroEventosParcela** [113] (véase extracto de código 6.17), y está diseñado para registrar eventos en la blockchain, proporcionando una capa de transparencia e inmutabilidad. Este contrato utiliza **Solidity** y está configurado para ejecutarse en la versión 0.8.29, porque es la versión más utilizada recientemente [98]. Su funcionalidad principal es emitir `EventoRegistrado`, que almacena un hash único del evento (`hashEvento`), hash de la parcela asociada (`hashParcela`), y el punto en el tiempo en el que se hizo la transacción o `timestamp`. Esto permite que los datos de los eventos sean registrados de manera confiable y accesible para su posterior consulta.

Además, este implementa un mecanismo de **control de acceso** que restringe la ejecución del mismo únicamente al propietario del contrato, que es la dirección que despliega el contrato en el caso de AgroFind es típicamente el relayer (véase 5.2.3) que actúa como intermediario para registrar los eventos en la blockchain.

```
/**
* Submitted for verification at amoy.polygonscan.com on 2025-04-03
*/
// AgroFind. Trabajo de Fin de Grado, hecho por Adrian Mena Ruiz.
// AgroFind. Final Degree Proyect, done by Adrian Mena Ruiz.
pragma solidity ^0.8.29;
contract RegistroEventosParcela {
    event EventoRegistrado(
        bytes32 indexed hashEvento,
        bytes32 indexed hashParcela,
        uint256 timestamp
    );
    // el relayer (aws lambda)
    address public owner;
    modifier onlyOwner() {
        require(msg.sender == owner, "Solo el owner");
        _;
    }
    constructor() { owner = msg.sender; }
    function registrarEvento(
        bytes32 _hashEvento, bytes32 _hashParcela
    ) external onlyOwner {
        emit EventoRegistrado(
            _hashEvento, _hashParcela, block.timestamp
        );
    }
}
```

**Extracto de código 6.17:** *Código del smart contract RegistroEventosParcela.*

[98] Esta versión se lanzó el 12 de marzo de 2025, y es la más actualizada respecto a la seguridad de los smart contracts. Fuente: [114].

### PolygonScan

Para poder ver en la blockchain de Polygon Amoy los eventos que se vayan registrando en el smart contract, se usa el buscador de **PolygonScan**, donde dada la dirección del contrato, se pueden ver todas las transacciones que suceden:

amoy.polygonscan.com/address/0x443a1fa4bf2a8b70529dda692518889cc45ff5d1

En la figura 6.44 se puede ver la lista de todas las veces que se ha invocado a RegistroEventosParcela, que ha llevado a transacciones en la blockchain. Si se hace clic en una de ellas, se pueden ver más detalles (figura 6.45), entre los que destacan las dos variables `hashEvento` y `hashParcela`, que son los mismos hashes que aparecen en el informe de la sección 6.2.2.

Esto es lo que verifica que efectivamente el evento está en la blockchain, y se puede ver también la hora exacta en la que se realizó. Además, existe más información por cada transacción, cómo por ejemplo el número de bloque en el que se ha ejecutado, el precio del *gas* (véase 3.3.2), el hash único de cada transacción, etc.

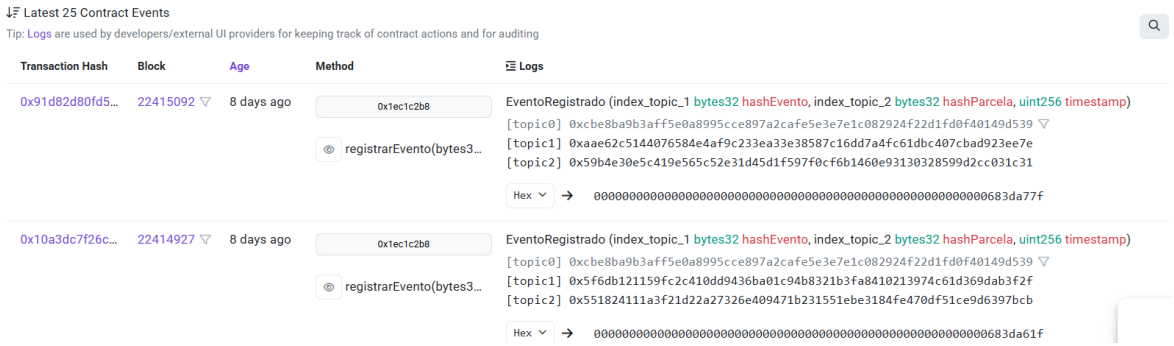



**Figura 6.44:** *Vista de eventos registrados en el contrato. Fuente: [113]*

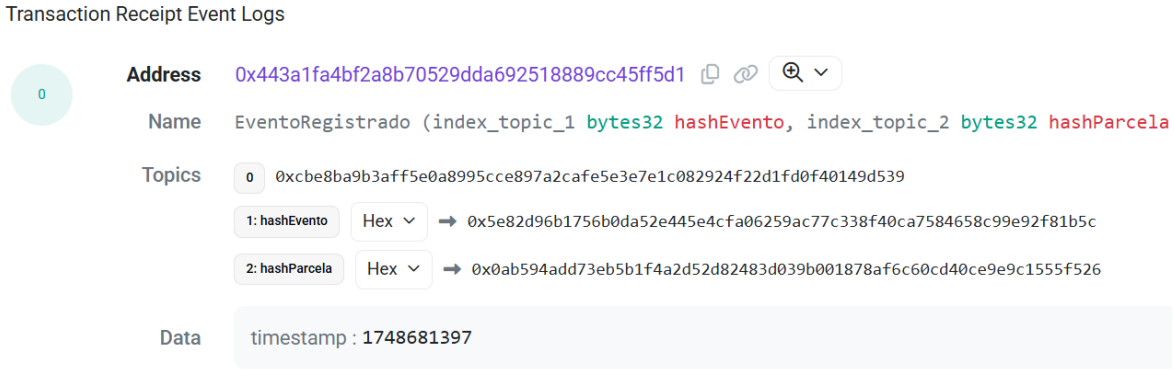



**Figura 6.45:** *Detalles de un evento concreto, pestaña logs. Fuente:* https://amoy.polygonscan.com/tx/0x21cf67d2c138c1fcc9da043f667a087dde80dba0b6272dbc4e40e4dfbe4f5aac

# 7. Pruebas

Toda aplicación ha de ir acompañada de un proceso que permita validar su correcto funcionamiento. En este apartado se recogen las pruebas a las que ha sido sometida AgroFind, donde se han realizado varios tipos distintos de validaciones (técnicas, multiplataforma y de usuario), sin embargo, en ocasiones se detectan casos especiales que han sido pasados por alto y han de ser corregidos a futuro.

## 7.1. Pruebas técnicas

### 7.1.1. Unitarias

Una **prueba unitaria** es una metodología de prueba de software que se enfoca en verificar la funcionalidad de las unidades más pequeñas y aisladas de código de una aplicación. Según el artículo de SmartBear [115], una *unidad* puede ser una función, un método o una clase específica. El objetivo principal es asegurar que cada componente individual de la aplicación funcione correctamente en solitario, facilitando la detección temprana de errores y simplificando el mantenimiento del código a posteriori. Las pruebas unitarias deben ser *rápidas, automatizadas* y *reproducibles*, sin depender de factores externos (véase [116]).

#### Configuración

**Mockito** es un *framework* de *mocking* o imitación muy popular en el ecosistema de Java/Kotlin que ha sido portado a Dart y es ampliamente utilizado en Flutter. En AgroFind se ha utilizado Mockito para realizar los tests porque permite crear pruebas aisladas, legibles y robustas al simular de manera efectiva el comportamiento de las dependencias, asegurando que cada unidad de código funcione como se espera.

Aún así, se han desarrollado las validaciones más críticas para asegurar el funcionamiento al máximo posible. Para realizar los *tests* se ha creado una carpeta `tests/` dónde se han codificado varios archivos `.dart` con las pruebas correspondientes.

- **`login_test.dart`**. Se encarga de imitar el ecosistema de Firebase para realizar pruebas sobre cómo funciona el inicio de sesión.
- **`registro_test.dart`**. Funciona de manera similar a la prueba de login, pero para el formulario del registro.
- **`eventos_test.dart`**. Su función es comprobar que los eventos se cargan visualmente de manera correcta entre los que se encuentran las fechas en su formato, los iconos y su color dependiendo del tema de la aplicación, qué mensaje muestra cuando no hay eventos...

- **`tiempo_test.dart`**. Se encarga de comprobar que la pantalla de tiempo es funcional al 100 %, que el botón de la ubicación actual si no se detecta ubicación no de errores, que el tiempo cargue los iconos correspondientes y que la temperatura en cada parcela sea la que corresponde.

Como ejemplo de cómo funcionan estos tests, se incluye un extracto de código de ellos, donde en el primero (7.1) se puede ver el uso de la clase **Mock**, que permite emular un servicio como Firebase para obtener resultados específicos (véase [117]).

```
/* imports varios */
class MockUser extends Mock implements User {}
class MockFirebaseAuth extends Mock implements FirebaseAuth {}

void main() {
  // asegurarse de que esta bien inicializado
  TestWidgetsFlutterBinding.ensureInitialized();
  /* codigo intermedio */
  testWidgets('parcelas del usuario', (WidgetTester tester) async {
  final mockAuth = MockFirebaseAuth();
  final mockUser = MockUser();
  when(mockAuth.currentUser).thenReturn(mockUser);
  /* resto de código */
```

**Extracto de código 7.1:** *Ejemplo de uso de Mockito y las clases tipo Mock, test de propias parcelas.*

La otra funcionalidad principal de Mockito es la sentencia **`expect`**, que es es el equivalente a los `assert` en JUnits o similares. Esto es lo que confirma al test lo que debe o no debe de aparecer dado un recurso, y para conocerlo en profundidad es recomendable ver el video de Youtube de *Unit Test with Mockito in Flutter* [118], ya que explica en detalle todo lo relacionado con esta sentencia, así cómo el extracto de código 7.2.

```
/* codigo anterior */
testWidgets('elementos en pantalla', (WidgetTester tester) async {
      await tester.pumpWidget(createLoginWidget());
      // elementos de texto
      expect(find.text('Correo electrónico'), findsOneWidget);
      expect(find.text('Contraseña'), findsOneWidget);
      expect(find.text('  Olvidaste  tu contraseña?'), findsOneWidget);
      expect(find.text('Iniciar Sesión'), findsOneWidget);
      expect(find.text(' No  te has registrado?'), findsOneWidget);
      expect(find.text(' Regístrate aquí'), findsOneWidget);
      // widgets
      expect(find.byType(LoginTextBox), findsNWidgets(2));
      expect(find.byType(BotonLogin), findsOneWidget);
      expect(find.byType(Image), findsOneWidget);
    });
/* resto de código */
```

**Extracto de código 7.2:** *Uso de sentencia expect, test de registro.*

Para ejecutar estas pruebas en Flutter es muy sencillo, ya que solo hay que utilizar el comando:

```
flutter test
```

Y si todos los archivos de pruebas están en el directorio `test/`, se ejecutarán uno tras otro. Los resultados de estos aparecen en la terminal, y se puede ver si han sido satisfactorios o erróneos.

### Resultados

Al ejecutar las pruebas de los cuatro archivos, se ha obtenido que 38 de los 40 tests han sido correctos.

```
PS C:\Users\User\Repositorios\AgroFind\agrofind> flutter test
......
00:25 +38 -2: Some tests failed.
```

Los dos errores han sido debido a la base de datos SQFlite, ya que como los tests se ejecutan en paralelo, la base de datos no está preparada para usar transacciones y evitar los bloqueos entre filas. Esto se debe principalmente a la interacción de cuando se obtienen los eventos, y es que se crean nuevos objetos `Evento` y posteriormente se les cambia la parcela a la correspondiente después de buscarla en la tabla. En la práctica al ejecutar la aplicación no da problemas, pero los tests lo detectan por **intentar acceder al mismo evento desde dos sitios distintos de manera simultánea en la misma ejecución**, y por tanto ocurre un interbloqueo:

```
Warning database has been locked for 0:00:00.500000. Make sure you always
use the transaction object for database operations during a transaction.
```

La solución a ello sería cambiar la manera de obtener los eventos, y hacer una función *lambda* nueva que devuelva las parcelas a la vez que los eventos, para no tener que añadir la parcela al evento después, cómo actualmente (extracto de código 7.3).

```
/* codigo anterior */
Future<List<Evento>> cargarEventosDesdeDynamo() async {
    try {
      final respuesta = await http.get(
        Uri.parse('${dotenv.env['API_URL_OBTENER_EVENTOS']}?
        idUsuario=${_usuarioActual!.uid}'),
        headers: {'x-api-key': dotenv.env['API_AGROFIND_KEY']!},
      );
      if (respuesta.statusCode == 200) {
        List<dynamic> info = json.decode(respuesta.body);
        for (var item in info) {
          try {
            Evento unEvento = Evento.fromJson(item);
            unEvento.parcela = await _servicioParcela.
              obtenerParcelaPorId(unEvento.idParcela);
/* resto de código */
```

**Extracto de código 7.3:** *Función cargarEventosDesdeDynamo, clase PaginaEventosPage.*

### 7.1.2. Automatizadas

Firebase Test Lab (FTL) es una herramienta de pruebas automatizadas que ofrece Google (véase [119]), y permite ejecutar pruebas en una gran variedad de dispositivos tanto físicos como virtuales, tanto para aplicaciones Android como iOS, subiendo el archivo `.apk` o `.ipa`. Entre sus funcionalidades se incluye la ejecución de pruebas unitarias (como *Espresso* o *XCTest*), así como pruebas automáticas como **Robo Test**, que es la que se ha usado en este proyecto.

**Robo Test** [120] es un tipo de prueba automatizada que no requiere escribir código, sino que su funcionamiento se basa en el análisis de la interfaz de la aplicación para recorrer automáticamente sus pantallas mediante eventos simulados como *clics*, *desplazamientos* y *entradas de texto*. Robo Test intenta explorar la mayor parte posible de la app, detectando **bloqueos** o **errores** durante la ejecución. Es especialmente útil para detectar fallos que no siempre se cubren con pruebas manuales o unitarias, y para poder ejecutar pruebas en varios dispositivos distintos.

#### Configuración

En la imagen 7.1 se pueden ver los parámetros que se han introducido previamente a realizar el análisis de Robo Test, y para ello se han seleccionado los siguientes tres dispositivos [99], donde todos ellos están configurados en orientación vertical y en Castellano:

- **Pixel 5** con API 30 (Android 11). Un teléfono con al menos 3 años, de gama baja/media que podría tener cualquier usuario, a punto de dejar de ser actualizado.
- **Pixel 3** con API 28 (Android 9). Dispositivo anticuado que ya no recibe actualizaciones, pero todavía podría ser utilizado por una fracción de los agricultores.
- **Small Phone 4.65in/12cm (Virtual)** con API 26 (Android 8.0). Emulador con medidas más pequeñas y con mayor antigüedad que el Pixel 3, prueba límite de la tecnología utilizada y con expectativas a posibles fallos.

[99] La aplicación ha sido probada manualmente en un emulador con API 33 (Android 13) y un teléfono físico con API 29 (Android 10), por esa razón se han elegido los dispositivos en el Robo Test, para cubrir lo máximo posible.

Además, se ha establecido un tiempo máximo de ejecución de **7 minutos** por prueba para evitar ejecuciones demasiado extensas. También se han proporcionado **credenciales de prueba**, permitiendo que el sistema pueda iniciar sesión dentro de la app. Para ello, se han definido los nombres de los recursos de usuario y contraseña (`login_correo` y `login_contrasena`) junto con sus respectivos valores de prueba que se han ido utilizando a lo largo de la memoria: `ejemplo@gmail.com` y `ejemplo`.

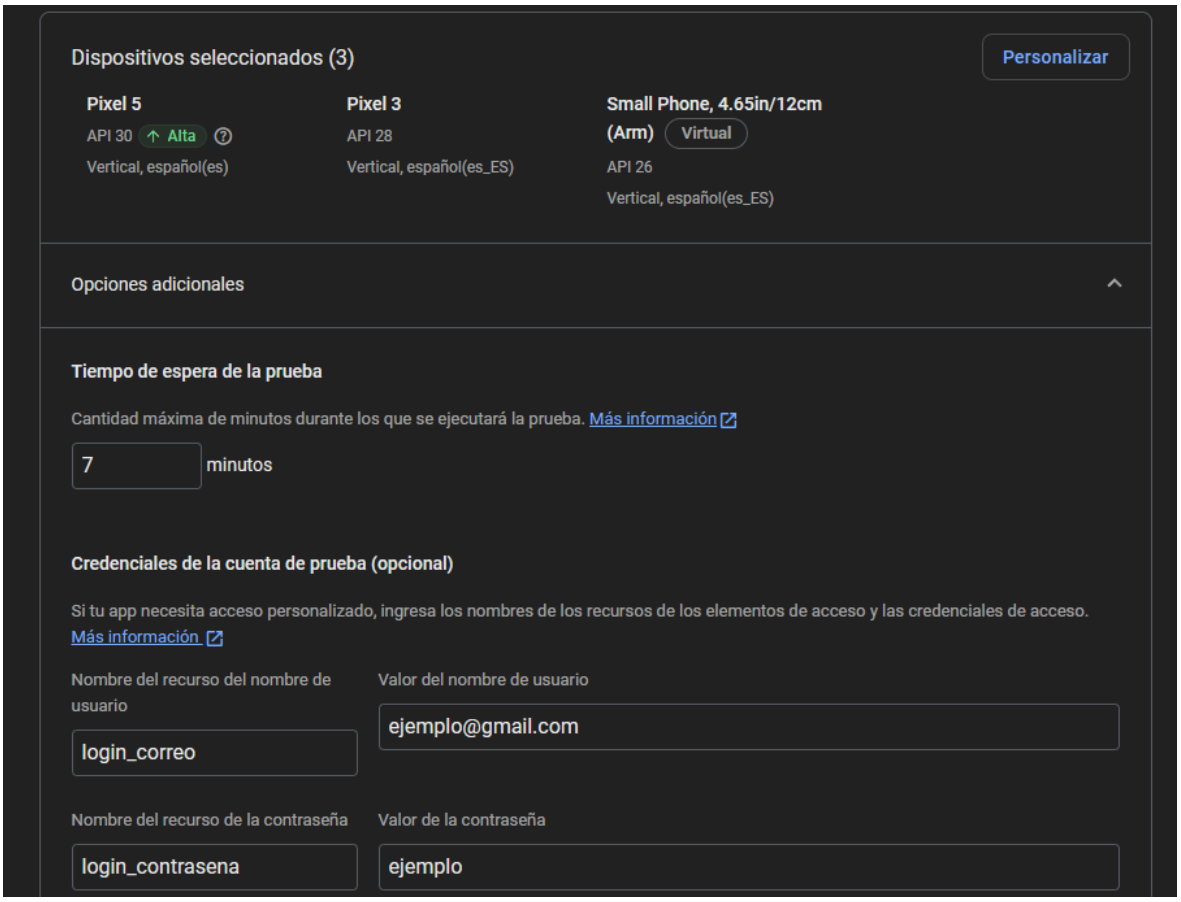



**Figura 7.1:** *Configuración inicial previa al Robo Test.*

### Resultados

Una vez iniciada la ejecución, hay que esperar varios minutos hasta su finalización y con ello se obtiene un informe muy completo (figura 7.2), llamado **matriz de resultados** sobre cómo han ido las ejecuciones en cada uno de los dispositivos. De manera inicial se puede ver que el dispositivo más antiguo ha dado algún fallo, y que los otros dos se han ejecutado correctamente.

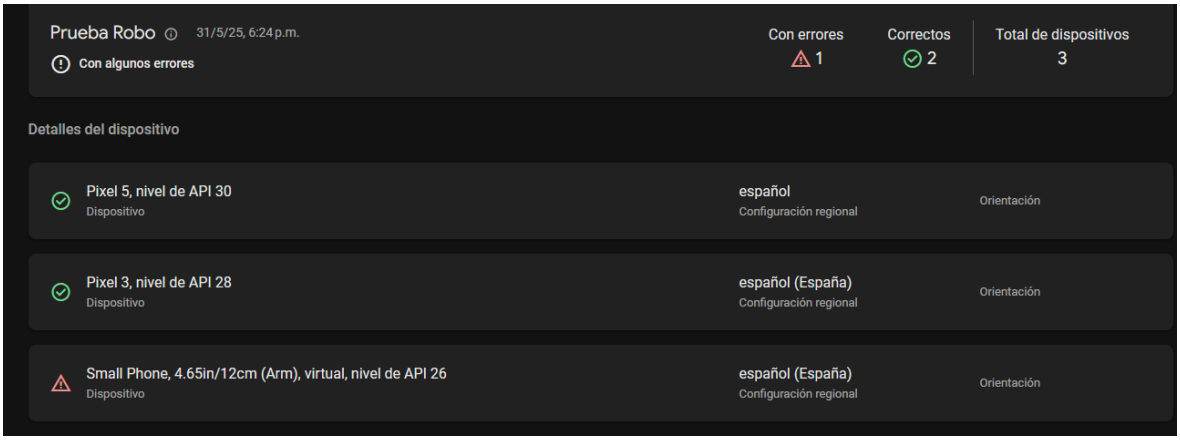



**Figura 7.2:** *Matriz de resultados al finalizar el Robo Test.*

Si analizamos uno a uno, podemos hacer una tabla con los resultados (7.1), además de incluir capturas de cada uno de ellos (véase figuras 7.3, 7.4 y 7.5) para certificar la veracidad de los mismos.

| | **Pixel 5** | **Pixel 3** | **Emulador Antiguo** |
|---|---|---|---|
| **API** | API 30 | API 28 | API 26 |
| **Versión Android** | Android 11 | Android 9 | Android 8 |
| **Duración** | 2 min 48 s | 4 min 19 s | 7 min 1 s |
| **Acciones realizadas** | 75 acciones | 107 acciones | 180 acciones |
| **Pantallas exploradas** | 42 pantallas | 42 pantallas | 1 pantalla |
| **¿Correcto?** | **Sí** | **Sí** | **No** |

**Tabla 7.1:** *Resumen de los resultados de la ejecución de Robo Test.*

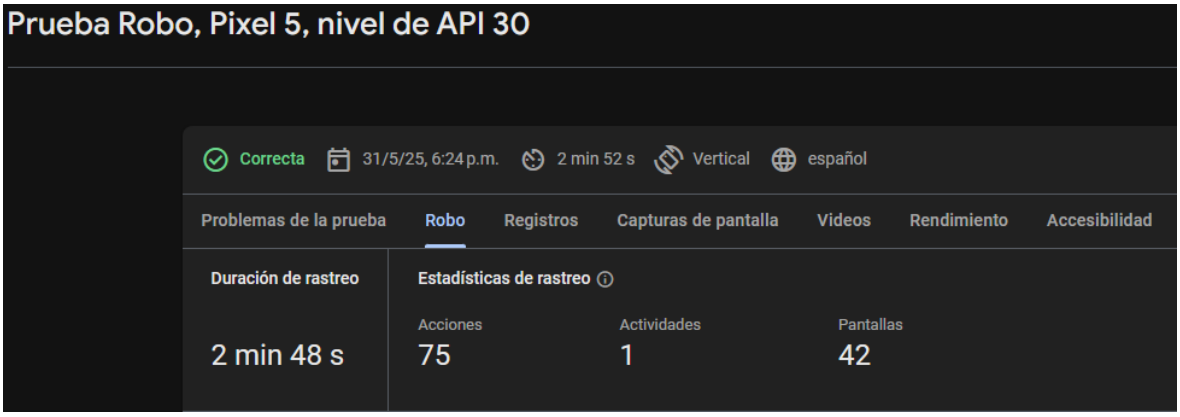



**Figura 7.3:** *Detalles de resultados del Pixel 5.*

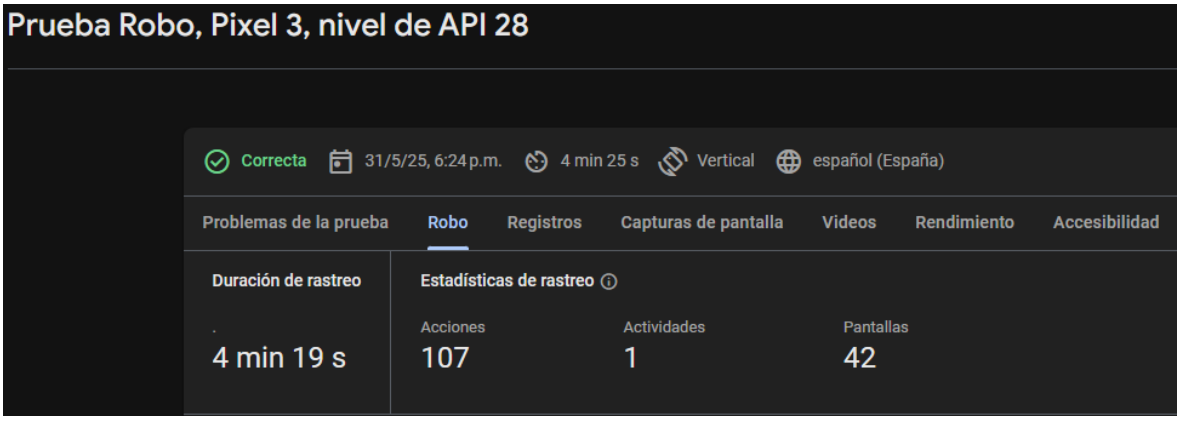



**Figura 7.4:** *Detalles de resultados del Pixel 3.*


**Figura 7.5:** *Detalles de resultados del teléfono emulado.*

En el caso del *Small Phone* o Emulador antiguo, tuvo errores y no pudo recopilar métricas completas de rendimiento. El hecho de que el tiempo de visualización completa no esté disponible (figura 7.6), junto con el estado *con errores*, sugiere que la aplicación no alcanzó un estado estable durante la prueba, y esto cuadra con el resultado de que solo haya pasado por una pantalla (probablemente al arrancar la aplicación).

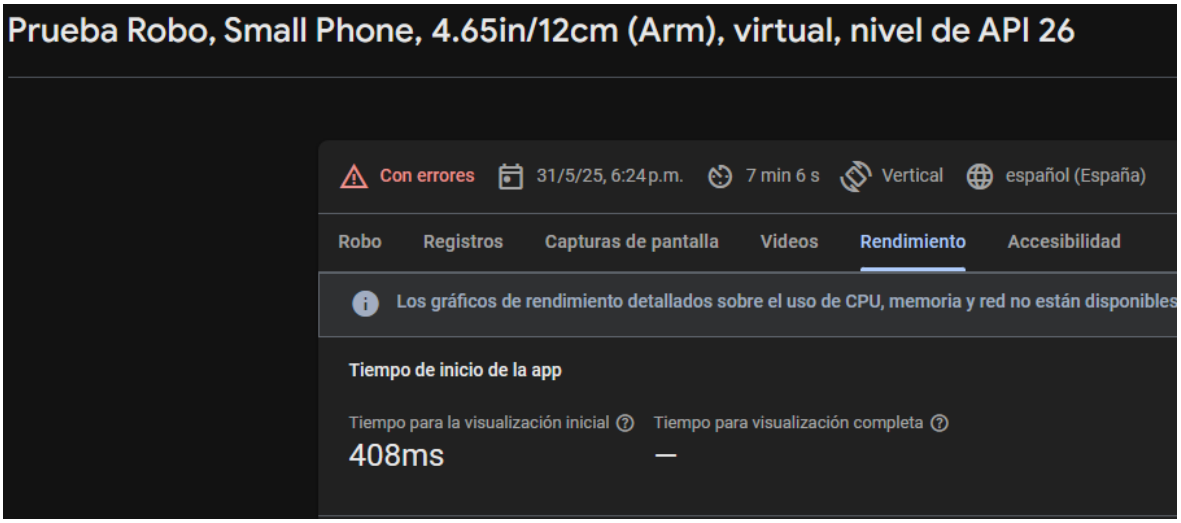



**Figura 7.6:** *Detección del error del teléfono emulado.*

Esto ha podido ser causado por varias razones, entre las que se encuentra que las librerías no sean compatibles con una versión tan antigua de Android, que la aplicación sea demasiado pesada para ejecutarla en un dispositivo con pocos recursos... Pero la causa real la encontramos en la figura 7.7, donde se ve claramente que alguna de las APIs utilizadas no pertenece al SDK oficial y por tanto esto puede tener dar pie a conflictos en versiones que no tengan el soporte a APIs externas en la SDK oficial de Android.

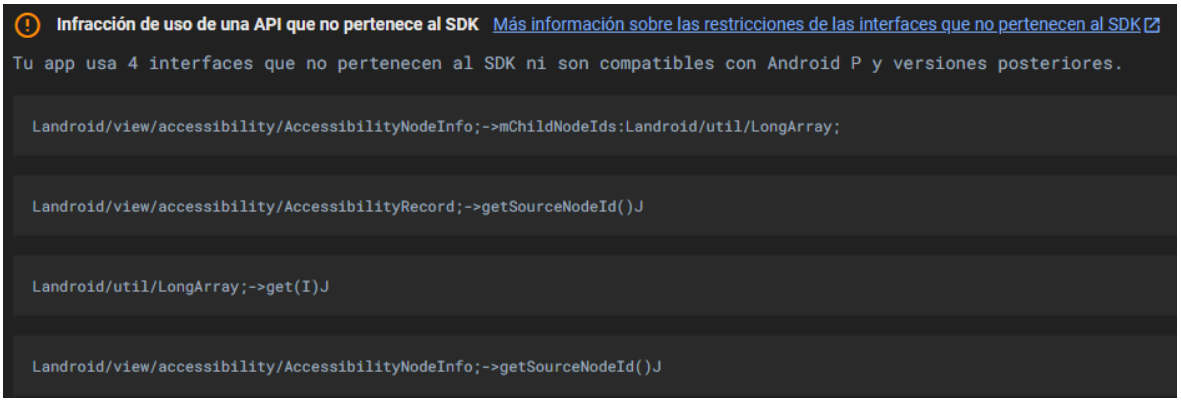



**Figura 7.7:** *Causa del error del teléfono emulado.*

### 7.1.3. Backend y APIs

Para hacer unas pruebas técnicas completas, es imprescindible comprobar que el backend funciona correctamente, sobre todo la conexión entre la aplicación con la API, que es lo que se analiza en esta sección.

#### Configuración

Para ello se puede hacer desde la propia consola de AWS cómo explican en la propia documentación para desarrolladores de API Gateway [121], pero en este caso, se ha exportado el archivo de definición de la AgroFindAPI (véase [122] y [123]) en formato **JSON, Open API 3** con las extensiones correspondientes para importarlo en **Postman** cómo se ve en la figura 7.8.

Una vez importado en la aplicación de escritorio, se pueden hacer pruebas, definiendo las variables necesarias como el *body* o cuerpo, las `headers` o cabeceras y añadiendo la clave privada, cómo se puede observar en la figura 7.9 (véase [124]), donde los métodos **OPT** son solo para la compatibilidad con navegador usando CORS.

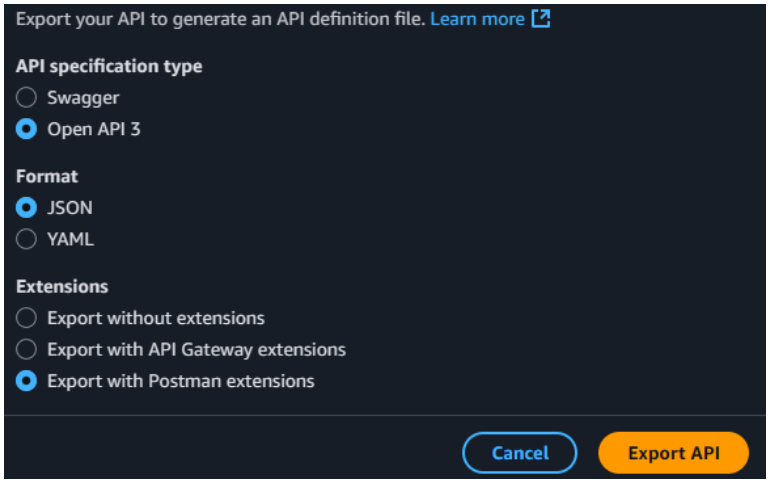



**Figura 7.8:** *Exportar definición de API en formatos distintos.*

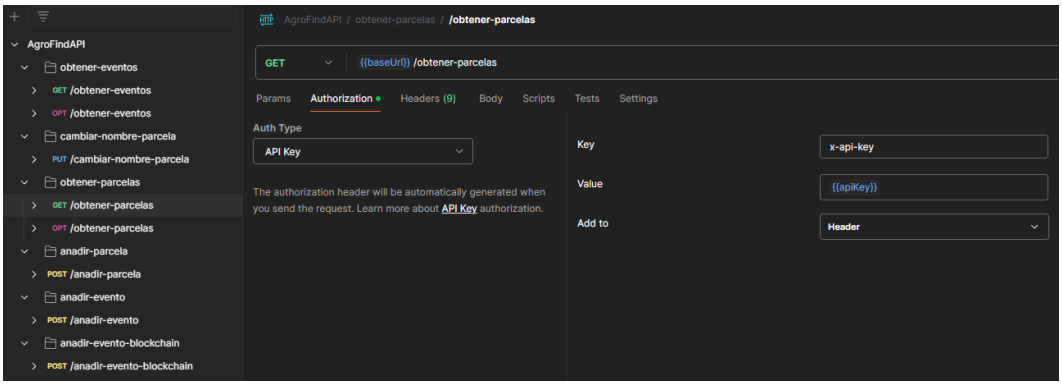



**Figura 7.9:** *Página desde la que se realizan las pruebas desde Postman.*

### Resultados

Para hacer todas las comprobaciones, se han creado varias tablas, una para cada *endpoint* de AgroFindAPI, donde se asume que en todas las tablas se envía como parámetro la clave de la API en la cabecera `x-api-key`, pero no se especifica para evitar repetirlo en todas las tablas posteriores. Además, la ruta base desde la que se accede a la misma es siempre la siguiente:
`https://6aa5blfxwk.execute-api.eu-south-2.amazonaws.com/develop`

La tabla 7.2 es la correspondiente a `/obtener-eventos`, a la que hay que pasarle un idUsuario, que devuelve la lista de eventos relacionados a ese usuario.

| **Ruta** | **Parámetro de ruta** | **Esperado** | **Resultado** |
|---|---|---|---|
| GET /obtener-eventos | `?idUsuario='Lma4cltEh4cu...'` | Lista de eventos, 200 OK | Lista de eventos, 200 OK |
| GET /obtener-eventos | `?idUsuario='idNoExistente'` | Lista vacía, 200 OK | Lista vacía, 200 OK |
| GET /obtener-eventos | — | 400 Bad Request | 400 Bad Request |

**Tabla 7.2:** *Pruebas: Obtener eventos.*

Siguiendo, la tabla 7.3 devuelve todas las parcelas existentes, esto se debe a que la aplicación debe de tener las coordenadas de los recintos de otros usuarios para que estas no puedan ser añadidas dos veces (aunque las Lambdas no lo permitan). Esto se hace para evitar errores internos en AgroFind.

| **Ruta** | **Cuerpo** | **Esperado** | **Resultado** |
|---|---|---|---|
| GET /obtener-parcelas | — | Lista de parcelas, 200 OK | Lista de parcelas, 200 OK |

**Tabla 7.3:** *Prueba AgroFindAPI: Obtener parcelas.*

Posteriormente se analizan las tablas 7.4 y 7.5, que funcionan de la misma manera en ambos casos. Son métodos POST que añaden un nuevo elemento a DynamoDB, que son los que se comprueban después de hacer las peticiones en el cliente Flutter. Nótese que no se incluye el cuerpo al completo en estos ejemplos, debido a que la lista de coordenadas y el WKT son extensos.

| Ruta | Cuerpo | Esperado | Resultado |
|---|---|---|---|
| POST /anadir-parcela | `Parcela no existente y con datos válidos {'datosParcela': { 'nombre': 'Terreno Sur', 'coordenadas': [[42.893772, -3.471678], [...,...], ...], 'informacion': {'WKT': ...} }, 'idParcela': 9214405053651, 'idUsuario': 'Lma4cltEh4cu...' }` | 200 OK | 200 OK |
| POST /anadir-parcela | `Parcela ya existente.` | Error 400 parcela existente. | Error 400 evento existente. |
| POST /anadir-parcela | `Parcela no existente, pero con algún dato inválido, por ejemplo: 'coordenadas': '42.893772'` | Error 406 al añadir parcela. | Error 406 al añadir parcela. |

**Tabla 7.4:** *Prueba AgroFindAPI: Añadir parcela.*

| Ruta | Cuerpo | Esperado | Resultado |
|---|---|---|---|
| POST /anadir-evento | `Evento no existente previamente: {'datosTipo': {'actividad': 'Cebada'}, 'tipo': 'Siembra', 'idParcela': 9214405053651, 'idUsuario': 'Lma4cltEh4cu...', 'fecha': 2025-06-09T09:59:40Z }` | 200 OK | 200 OK |
| POST /anadir-evento | `Evento ya existente en DynamoDB.` | Error 400 evento existente. | Error 400 evento existente. |
| POST /anadir-evento | `Evento sin todos los datos requeridos, por ejemplo: {'datosTipo': {'actividad': 'Aspersión'}, 'tipo': 'Riego' }` | Error 406 al añadir evento. | Error 406 al añadir evento. |

**Tabla 7.5:** *Prueba AgroFindAPI: Añadir evento.*

Posteriormente, se encuentra la tabla 7.6, que es una petición PUT que modifica el nombre de una parcela. Este es el caso en el que el usuario puede introducir alguna variable no controlada por el sistema, y por tanto se prueba más que el resto.

| **Ruta** | **Cuerpo** | **Esperado** | **Resultado** |
|---|---|---|---|
| PUT /cambiar-nombre-parcela | `Datos correctos en parcela existente.`<br>`{`<br>`'idParcela': '9214405053651',`<br>`'nuevoNombre': 'Terreno Sur 2',`<br>`'idUsuario': 'Lma4cltEh4cu...'`<br>`}` | 200 OK | 200 OK |
| PUT /cambiar-nombre-parcela | `Datos correctos en parcela inexistente.`<br>`{`<br>`'idParcela': 'idNoExistente',`<br>`'nuevoNombre': 'Terreno Nuevo',`<br>`'idUsuario': 'Lma4cltEh4cu...'`<br>`}` | Error 404 Not Found. | Error 404 Not Found. |
| PUT /cambiar-nombre-parcela | `Datos idUsuario incorrecto.`<br>`{`<br>`'idParcela': '9214405053651',`<br>`'nuevoNombre': 'Terreno Nuevo',`<br>`'idUsuario': 'idNoExistente'`<br>`}` | Error 404 Not Found. | Error 404 Not Found. |
| PUT /cambiar-nombre-parcela | `Sin nombre en la petición.`<br>`{`<br>`'idParcela': '9214405053651',`<br>`'idUsuario': 'Lma4cltEh4cu...'`<br>`}` | Error 406 al cambiar nombre. | Error 406 al cambiar nombre. |
| PUT /cambiar-nombre-parcela | `Nombre inválido en la petición. Puede ser null, algún carácter especial...`<br>`{`<br>`'idParcela': '9214405053651',`<br>`'nuevoNombre': null,`<br>`'idUsuario': 'Lma4cltEh4cu...'`<br>`}` | Error 406 al cambiar nombre. | Error 406 al cambiar nombre. |

**Tabla 7.6:** *Prueba AgroFindAPI: Cambiar nombre parcela.*

Finalmente, la tabla 7.7, en la que se analiza cómo se añade un evento a la blockchain. Esto varía si es una cosecha o no, porque en ese caso cambian los datos del evento. Hay que tener en cuenta que se duplica el idParcela para que la Lambda no necesite acceder a `datosEvento`, puesto que este tiene una estructura cambiante.

| Ruta | Cuerpo | Esperado | Resultado |
|---|---|---|---|
| POST /anadir-evento-blockchain | `Añadir eventoBlockchain no existente.`<br>`{ 'datosEvento':`<br>`{`<br>`'datosTipo': {`<br>`'actividad': 'Aspersión'`<br>`},`<br>`'idUsuario': 'Lma4cltEh4cu...',`<br>`'idParcela': 9214405053651,`<br>`'fecha': 2025-06-09T19:50:13Z,`<br>`'tipo': 'Riego'`<br>`}`<br>`'idParcela': 9214405053651`<br>`}` | 200 OK | 200 OK |
| POST /anadir-evento-blockchain | `Añadir eventoBlockchain faltando información relevante. Ejemplo:`<br>`{`<br>`'idParcela': 9214405053651`<br>`}` | Error 406 al registrar evento. | Error 406 al registrar evento. |
| POST /anadir-evento-blockchain | `EventoBlockchain correcto pero algún error al añadir el evento a la blockchain.` | Error 406 al registrar evento. | Error 406 al registrar evento. |

**Tabla 7.7:** *Prueba AgroFindAPI: Añadir evento blockchain.*

Es importante mencionar que también se han realizado sendas pruebas individualmente en cada Lambda, utilizando la herramienta de validación propia de AWS [125], donde se comprueba el funcionamiento con un evento de prueba (figura 7.10).


**Figura 7.10:** *Ejemplo de prueba en consola de AWS, Lambda RelayerAgroFind.*

## 7.2. Pruebas multiplataforma

Al estar desarrollando en un framework como Flutter, una de las mayores ventajas que tiene es el poder hacer un solo código para las plataformas más comunes, pero no es tan sencillo como codificarlo para una de ellas y que sea compatible con el resto, sino que hay ciertos ajustes que hay que tener en cuenta. Para ello se comprobarán 3 plataformas diferentes que son **Android**, **Web** (con varios navegadores) y **Windows**, pero esto deja varias preguntas en el aire: *¿Por qué no se prueba con macOS e iOS? ¿Y con Linux?* Todas ellas se responderán en esta sección.

### 7.2.1. Dispositivos móviles

Todas las capturas de la interfaz de usuario de la aplicación que se han hecho en la sección 6.1.4 han sido en Android, y estas verifican que AgroFind funciona sin ningún fallo en este sistema operativo (a falta de las pruebas de usuario, véase 7.3).

#### Android

En la sección anterior se han hecho pruebas en Robo Test con distintos dispositivos, todos Android, pero además de ello, se ha probado en otros tres dispositivos más, que han sido los que ha empleado el autor para el desarrollo de la aplicación, y por tanto se hará un breve resumen de los resultados obtenidos:

Primeramente, en general se han obtenido resultados muy satisfactorios. Todos los dispositivos ejecutaron la aplicación correctamente, con excepción del emulador con Android 8.0 (API 26), que presentó problemas no reproducidos en otros entornos. La lista de todos los dispositivos empleados y sus peculiaridades es la siguiente:

- **Xiaomi 11 Lite 5G NE**, Android 13. Es el dispositivo físico más moderno que se ha utilizado en las pruebas de usuario (véase 7.3), y no ha dado ningún tipo de problema. Las dimensiones han sido las correctas en todo momento y el funcionamiento de la aplicación ha sido satisfactoria.
- **Emulador Moderno**, Android 13. Este emulador es el asociado a todo el desarrollo, en el que se ha probado un dispositivo emulado y otro físico.
- **Oppo A56 5G**, Android 12. Segundo dispositivo físico utilizado, que ha completado las pruebas de usuario (véase 7.3). Los márgenes horizontales no coinciden del todo, pero no afecta a la funcionalidad.
- **Pixel 5**, Android 11. Uno de los utilizados en las pruebas de Robo Test.
- **Xiaomi Mi A2 Lite**. Android 10. El teléfono físico que se ha usado junto con el emulador durante el desarrollo. Este ha sido el que se ha probado de manera mas exhaustiva.
- **Pixel 3**, Android 9. Uno de los utilizados en las pruebas de Robo Test.

- **Emulador Antiguo**, Android 8.0. Único dispositivo Android en el que no se han realizado pruebas exitosas.

### iOS

Hay que tener en cuenta que no se han realizado pruebas concluyentes en dispositivos iOS, por lo que el comportamiento de la aplicación en este sistema operativo no está garantizado. Esto se debe a que el tiempo para desarrollar las pruebas ha sido limitado, y al no tener un dispositivo físico iOS, habría que haber utilizado alguna herramienta como **Codemagic** [100], de manera completa pero se ha decidido que esto sea trabajo a futuro, debido que se ha intentado utilizar pero se ha desechado la idea.

Lo que se ha realizado ha sido crear un workflow de Codemagic usando la documentación que da la propia página web para hacer ejecuciones en proyectos Flutter (véase [126]) personalizando un archivo `codemagic.yaml` (véase extracto de código 7.4 y figura 7.11) para hacer una ejecución en un dispositivo iOS, a partir del repositorio de Github donde está alojado el proyecto.

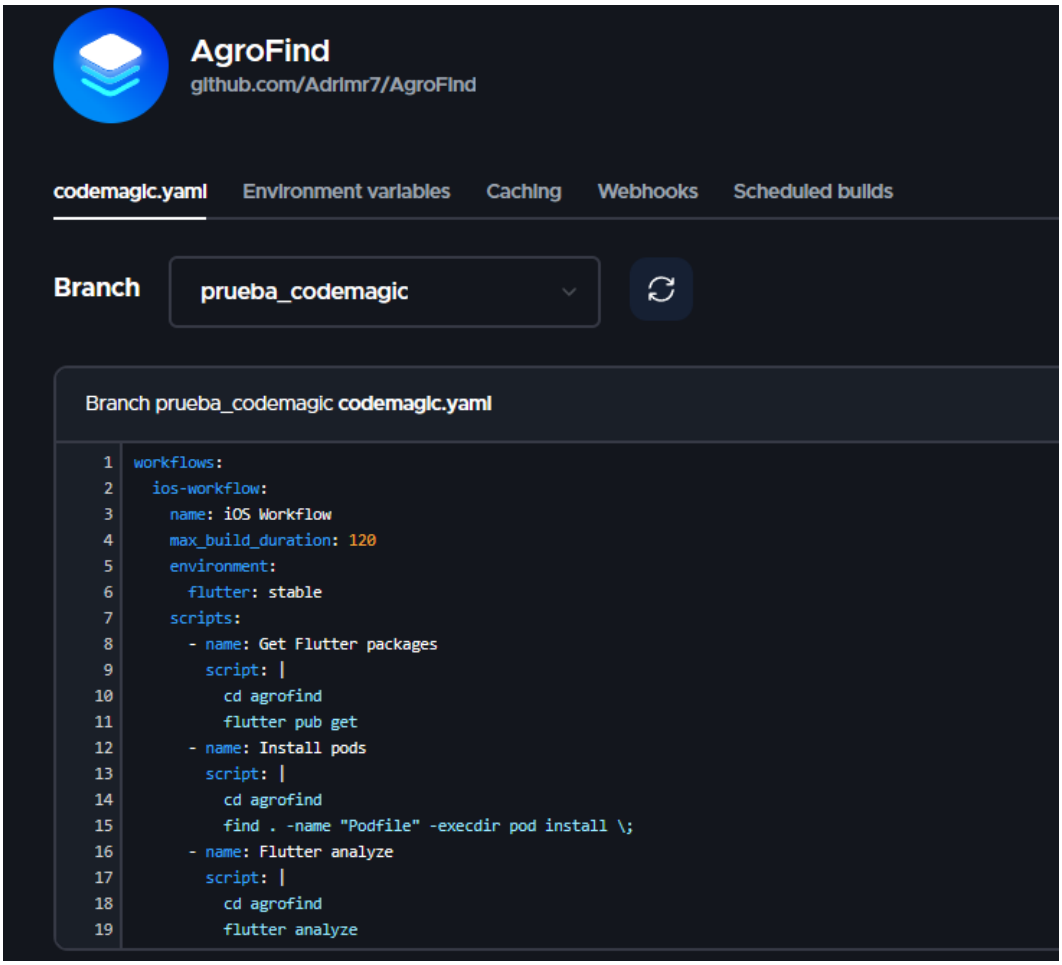



**Figura 7.11:** *Configuración inicial de Codemagic.*

[100]Página oficial de Codemagic: https://codemagic.io/start/.

Esto ha llevado a una ejecución del workflow, que ha sido errónea al analizar el proyecto (figura 7.12), debido a que no se han tenido en cuenta ciertos ajustes necesarios, pero cómo se ha explicado, continuar con las pruebas sería trabajo a futuro.

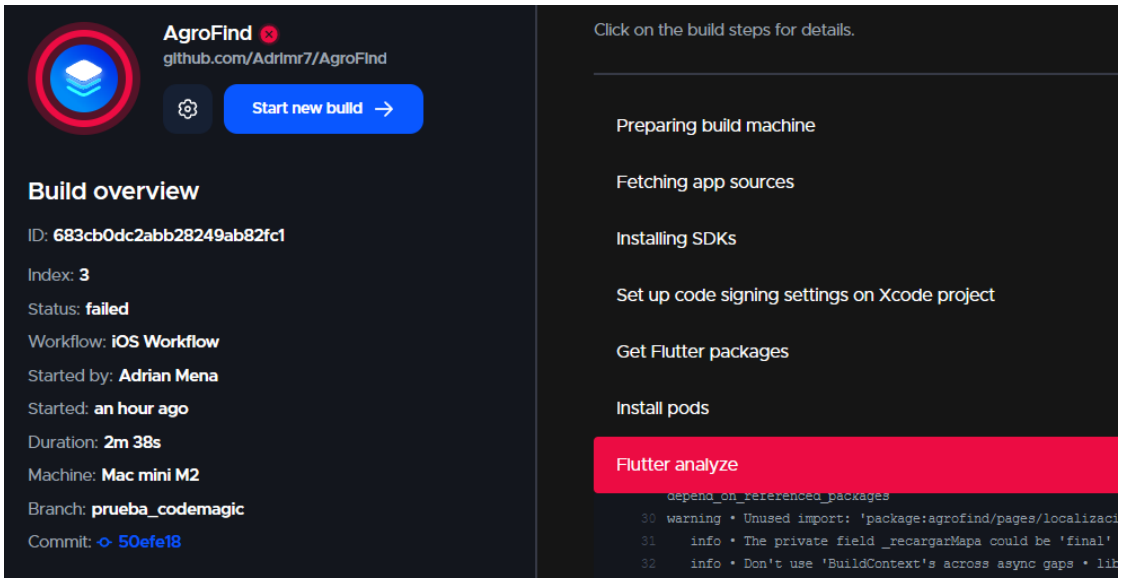



**Figura 7.12:** *Resultados de ejecución en Codemagic.*

```yaml
workflows:
  ios-workflow:
    name: iOS Workflow
    max_build_duration: 120
    environment:
      flutter: stable
    scripts:
      - name: Set up code signing settings on Xcode project
        script: |
          xcode-project use-profiles
      - name: Get Flutter packages
        script: |
          cd agrofind
          flutter pub get
      - name: Install pods
        script: |
          cd agrofind
          find . -name "Podfile" -execdir pod install \;
      - name: Flutter analyze
        script: |
          cd agrofind
          flutter analyze
      - name: Flutter build ipa
        script: |
          cd agrofind
          flutter build ipa --release
    artifacts:
      - build/ios/ipa/*.ipa
      - /tmp/xcodebuild_logs/*.log
```

**Extracto de código 7.4:** *Archivo codemagic.yaml utilizado para el workflow.*

### 7.2.2. Web

Al ser una aplicación diseñada para ser multiplataforma, hay que realizar pruebas en más plataformas, entre las que tendría más sentido funcional es en navegador [101]. Para determinar si la aplicación se está ejecutando en entorno web, se ha utilizado la constante `kIsWeb` del paquete `foundation` de Flutter. Una vez hecho eso, para poder probar en navegadores, hay que ejecutar unos comandos desde la raíz del proyecto para construir el artefacto web. Después hay que levantar un servidor rápido en local usando `http.server`.

```
flutter build web
python -m http.server --directory build/web
```

Finalmente, se abre cualquier navegador en el puerto 8000 en local (`localhost:8000`) para poder interactuar con AgroFind.

Al usar *SQFlite* como base de datos local (que es muy comúnmente utilizada en Flutter), se da la casuística de que no es compatible con plataformas web. Debido a esto, para que la aplicación se pueda utilizar en le web, se ha optado por cargar los datos de forma dinámica cada vez que el usuario accede a ellos, en lugar de almacenarlos localmente. Hay que decir que se está barajando desde los creadores de *SQFLite* realizar una versión portable para versiones de navegador cómo se menciona en sus issues [127], pero todavía no se ha llevado a cabo.

Esto también ha implicado gestionar correctamente las políticas *CORS* [102] al consumir servicios web, y más concretamente con la AgroFindAPI, que al ser personalizada no permitía por defecto estas políticas y ha habido que hacer el cambio posteriormente.

[101] Es importante mencionar que se ha probado en 3 navegadores distintos. Brave y Chrome (ambos basados en Chromium) por un lado y Firefox (basado en Gecko) por el otro. Todas las pruebas han sido similares, con un matiz que se menciona en la siguiente página.

[102] Es un mecanismo basado en cabeceras HTTP que permite a un servidor indicar cualquier dominio, o puerto con un origen distinto del suyo desde el que un navegador debería permitir la carga de recursos. Fuente: https://developer.mozilla.org/es/docs/Web/HTTP/Guides/CORS.

Para ello, se han seguido tanto la guía oficial de Amazon sobre cómo habilitar CORS desde la consola para API Gateway (véase [128]) como un *troubleshooting* de algunos errores que se han dado en el proceso [129].

Respecto a la funcionalidad, los procesos de inicio de sesión y registro funcionan al completo, al igual que el cambio de tema y la obtención del tiempo (figura 7.13). Los mapas funcionan correctamente, sin embargo, los eventos se cargan sin nombre por la manera en la que se hizo desde un primer momento toda la infraestructura y no se puede acceder al detalle de los eventos, ya que esto requeriría una reestructuración de la base de datos y cambios en el modo de acceso a ella.

Por tanto, la versión web podría ser útil para ver los eventos (figura 7.14), el mapa y el tiempo, pero no permite añadir nuevos eventos y por tanto *no es funcional al 100 %*. Para solucionarlo habría que hacer una transición a otro tipo de base de datos que sea compatible con navegadores, cómo se explicará en las conclusiones.

Finalmente, hay que mencionar que el gradiente superior de la `AppBar` no se ha escalado correctamente a las dimensiones distintas del navegador, y esto se debe a la manera peculiar en la que funciona el gradiente, cómo se ha explicado con la *curva de Bézier*. Para solucionar esto habría que hacer la curva de manera proporcional al tamaño de la pantalla, puesto que se han tenido en cuenta ciertos anchos pero no como un navegador.

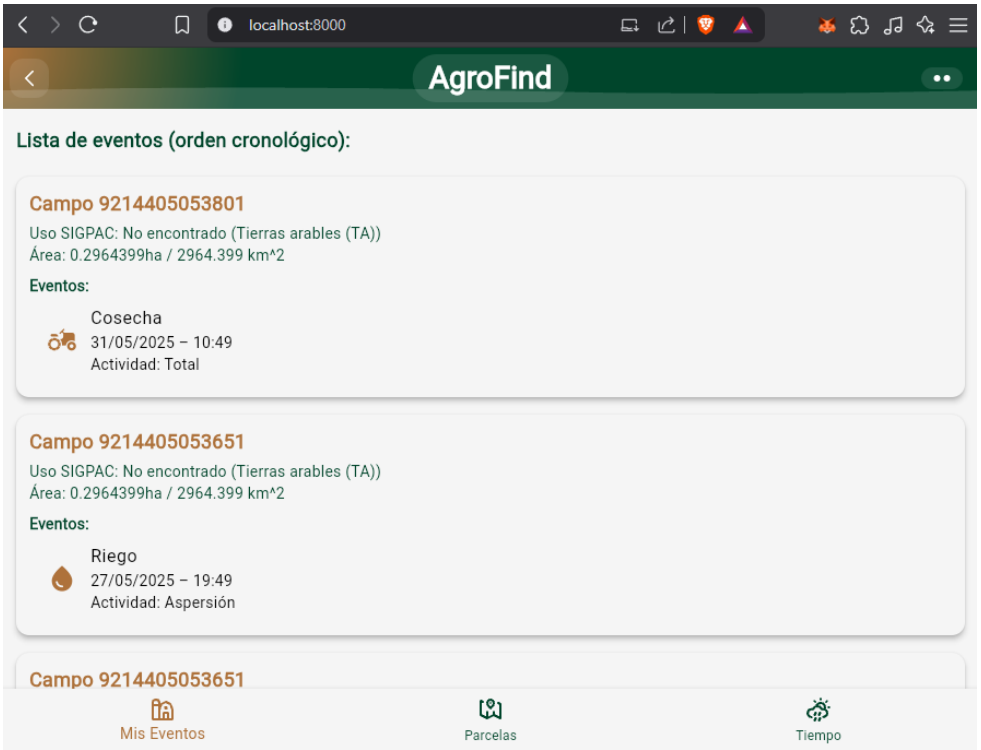



**Figura 7.13:** *Captura de pantalla, página de eventos en el navegador.*

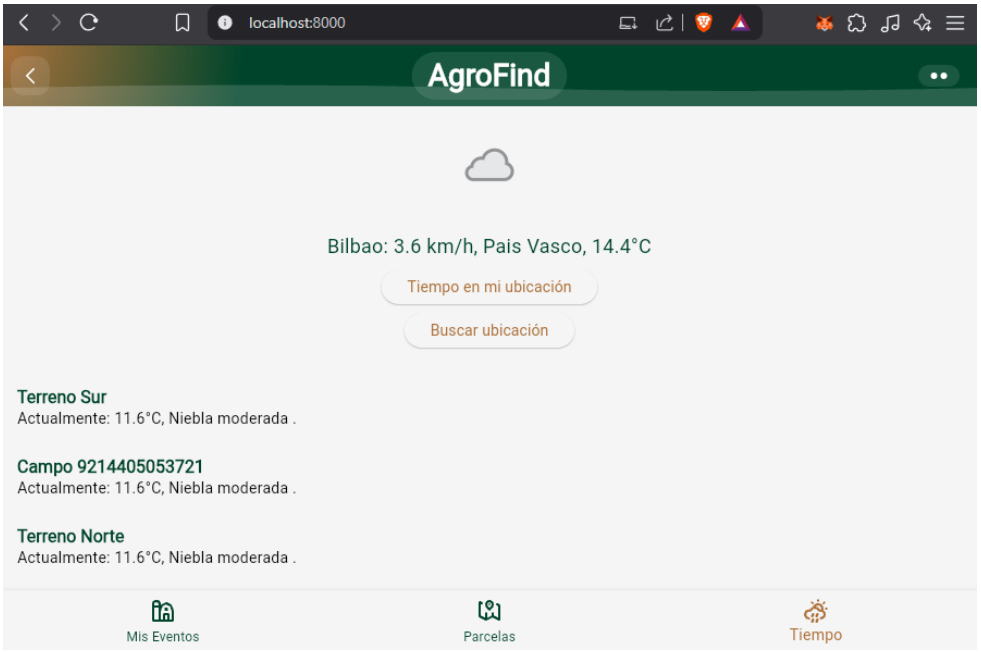



**Figura 7.14:** *Captura de pantalla, página del tiempo en el navegador.*

Nota: al renderizar los archivos SVG en Firefox, estos tardan unos instantes más, y esto se debe a que no se ha distinguido en el desarrollo entre los navegadores, y no se ha utilizado el motor gráfico de Firefox. Esto sería una mejora a futuro en caso de que se considere utilizar la app en la web.

### 7.2.3. Windows

Para que la aplicación sea realmente multiplataforma, es necesario comprobarlo en escritorio, y en este caso se ha utilizado Windows para hacer las pruebas.

Cómo trabajo futuro habría que hacer las pruebas tambien en Linux y macOS, aunque el objetivo de usuarios de la aplicación no usa Linux y por tanto no tiene tanta importancia como si la puede tener el sistema operativo de Microsoft. Hacer unas pruebas en macOS sí sería útil, porqué pueden haber algunos agricultores que prefieran todo el ecosistema Apple y tengan un portátil o un ordenador de sobremesa, pero al igual que con iOS [103], se propone como tareas para hacer si la aplicación se pusiera en producción y no como la prueba de concepto que es.

Poder compilar la aplicación en Windows (véase [130]) requiere de tener instalado Visual Studio (no confundir con Visual Studio Code), incluyendo las herramientas de desarrollo de C++ para escritorio. Este entorno es obligatorio ya que Flutter necesita compilar código nativo, y esto lo hace traduciendo Dart a C++.

[103]Nótese que se ha planteado previamente cómo hacer las pruebas para iOS, y se podrían hacer de manera similar para macOS, usando Codemagic, se reitera que esto sería un posible trabajo a futuro.

Además, antes de compilar, se recomienda ejecutar `flutter doctor` para verificar que todas las dependencias están correctamente configuradas, porque puede faltar algún aspecto.

De una manera similar a todo lo previo, se compila usando `flutter build windows`, y se genera un ejecutable en la ruta `build/windows/runner/Release`. Este ejecutable .exe tiene que ser ejecutado con las dependencias que tiene en la propia carpeta `Release` (figura 7.15) y por tanto esa carpeta se comprime y se podría llevar a cualquier dispositivo.

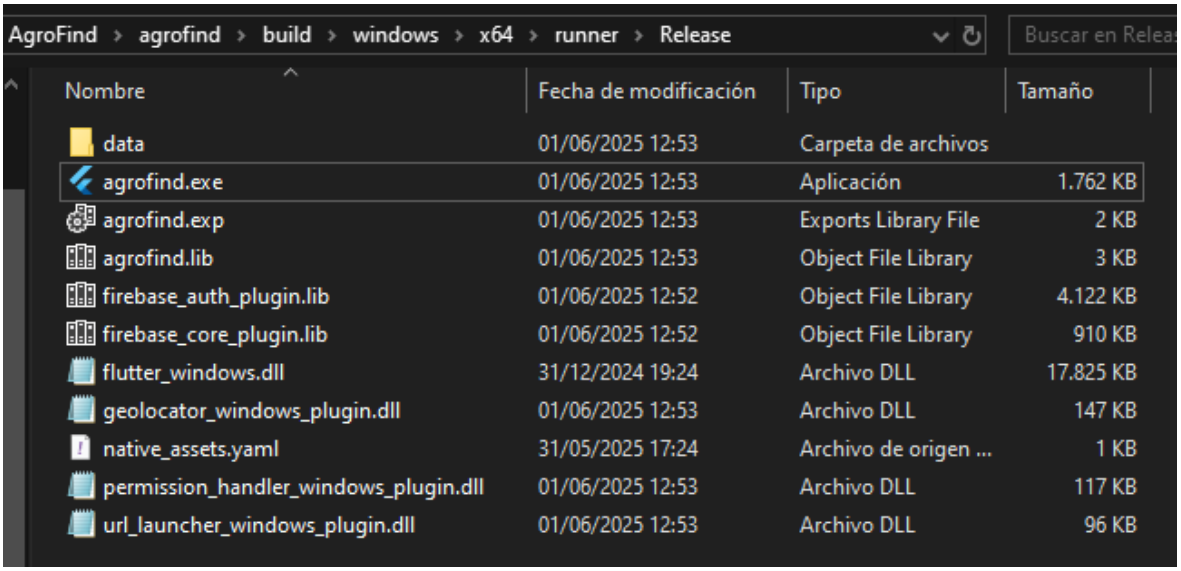



**Figura 7.15:** *Dependencias del ejecutable en Windows.*

Al ejecutar el archivo, existen varios problemas que se detallan en esta lista:

- Los eventos cargan, pero no se pueden utilizar para nada, debido a que la librería `SQFlite` no es compatible con Windows. Una posible solución sería utilizar otra librería en el caso de aplicaciones de escritorio como pueden ser `sqlite3`[104] o `drift`[105].
- Al intentar cargar los mapas, la aplicación deja de responder y se queda en blanco. Esto puede ser debido al uso de `TileProviders` ya que, cómo se explica en la documentación de `flutter_map` [131], en aplicaciones de escritorio se recomienda utilizar `AssetTileProvider` mientras que en el resto se usa `NetworkTileProvider`.

En este caso la prueba es insatisfactoria, debido a que al no poder acceder a los mapas (pese a poder iniciar sesión, véase figura 7.16), el propósito de AgroFind no se puede ejecutar correctamente.

[104]Paquete que proporciona acceso directo a bases de datos SQLite desde Dart sin depender de plataformas nativas. Ideal para aplicaciones Flutter de escritorio y CLI.
https://pub.dev/packages/sqlite3

[105]ORM (Object-Relational Mapper) para bases de datos en Flutter y Dart. Permite escribir consultas SQL de forma segura y tipada, y funciona con SQLite en escritorio, móvil y web.
https://pub.dev/packages/drift

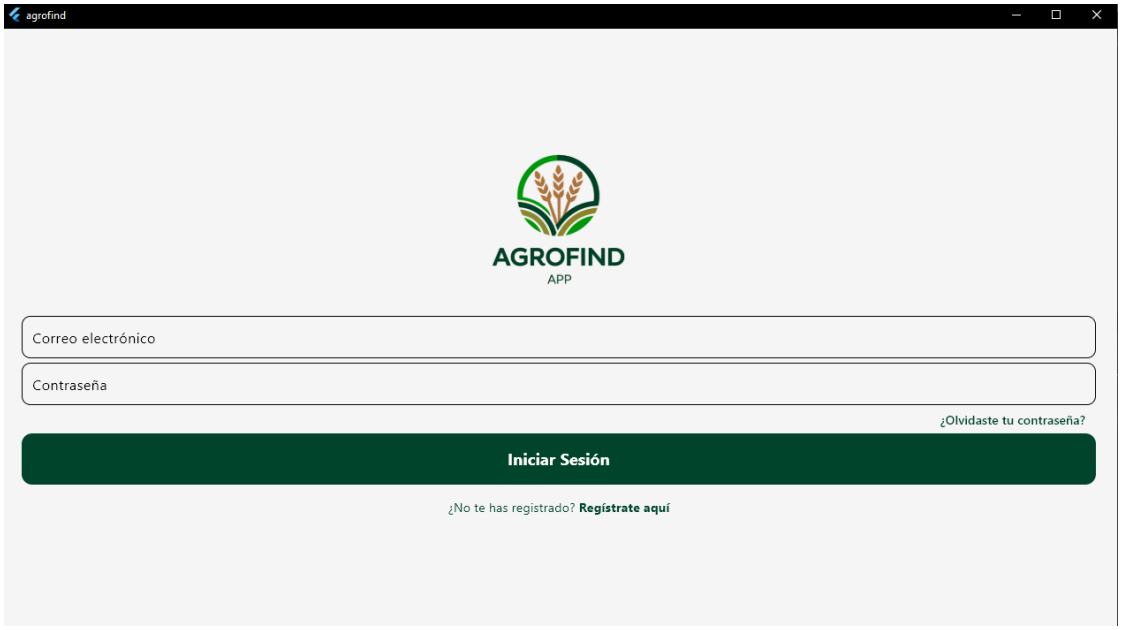



**Figura 7.16:** *Captura de pantalla, página de inicio de sesión en el navegador.*

## 7.3. Pruebas de usuario

En esta sección se abordan las pruebas más indispensables en una aplicación que va dirigida a un grupo de personas con un nivel de destreza medio o bajo en las nuevas tecnologías cómo son la mayoría de agricultores. Por tanto, los usuarios que han probado AgroFind en esta fase han sido en su mayoría personas con un conocimiento estándar sobre aplicaciones móviles, con los que se ha seguido una metodología concreta para obtener unos resultados lo más homogéneos posibles.

### 7.3.1. Metodología

- **Objetivo de las pruebas:** Evaluar la usabilidad de las funciones clave de la app AgroFind por parte de usuarios con conocimientos básicos o medios en el uso de tecnologías móviles. Determinar que partes de la aplicación pueden ser mejorables respecto a la interfaz y en caso de que sea así, encontrar posibles fallos.
- **Perfil de los participantes:**
  - Total de participantes: 5.
  - Edad: entre 32 y 78 años.
  - Nivel tecnológico: desde bajo hasta medio.
  - Ocupación: trabajos principalmente no relacionados con la informática, entre los que dos de los usuarios, que se dedican actualmente o se han dedicado en el pasado al mundo de la agricultura como profesión.
- **Entorno de la prueba:**
  - Dispositivos utilizados: Oppo A56 5G para un usuario, Xiaomi 11 Lite 5G NE para dos usuarios y Xiaomi Mi A2 para los otros dos restantes.
  - Asistencia: moderador presente solo para observación, sin intervención salvo exceso de tiempo en una tarea concreta.
- **Tareas propuestas:**
  1. Crear una cuenta en la aplicación, con correo, nombre y contraseña sin determinar previamente.
  2. Añadir una nueva parcela.
  3. Añadir un evento asociado a una parcela.
- **Criterios evaluados:**
  - Porcentaje de éxito por tarea.
  - Tiempo de ejecución de la misma (medio y máximo).
  - Nivel de satisfacción percibida (valorada del 1 al 10), teniendo en cuenta la dificultad, y la presencia de la interfaz.

### 7.3.2. Resultados

Una vez hecho el experimento en 3 sesiones, donde se han explicado a cada usuario de la misma manera las tareas y por separado, se han obtenido los siguientes resultados:

| Tarea | Éxito | Tiempo medio | Tiempo máximo | Nota |
|---|---|---|---|---|
| **Crear cuenta** | 100 % | 1 min 30 s | 2 min 3 s | 9 |
| **Añadir parcela** | 80 % | 1 min 56 s | No aplica | 7.5 |
| **Añadir evento** | 100 % | 1 min 12 s | 2 min 47 s | 8.5 |

**Tabla 7.8:** *Resumen de resultados de las pruebas de usuario en tareas clave.*

A continuación se detallan más concretamente todas las conclusiones y los puntos a mejorar tarea por tarea:

- **Crear cuenta**. Todos los usuarios han sido capaces de crearse una nueva cuenta, donde los que más han tardado ha sido por introducir la contraseña de manera errónea. Se ha encontrado un leve fallo al introducir un correo electrónico inválido, debido a que no apareció el mensaje avisando de cuál había sido el fallo (se ha solucionado rápidamente).
  Aún así, las opiniones han sido muy positivas y se ha sugerido introducir un botón para poder ver la contraseña después de haberla escrito.
- **Añadir parcela**. Esta ha sido la tarea en la que un usuario ha necesitado asistencia, pero que el resto no han tenido problemas. La queja de ese usuario ha sido que no está claro la diferencia entre *mis parcelas* y *parcelas SIGPAC*, y que habría que tener un botón para poder añadir una nueva parcela desde el menú de parcelas.
  El resto de usuarios cree que ha sido sencillo, pero el feedback más común es que podría ser mas intuitivo, entre lo que destaca la idea de añadir un tutorial la primera vez que se accede a la aplicación con una cuenta nueva.
- **Añadir evento**. Se ha podido completar en su plenitud y el feedback ha sido positivo de manera general. Un usuario sugiere que el botón de añadir evento sea más grande o tenga algo de texto en él, para poder verlo mejor.

#### Conclusiones

Las pruebas de usuario han sido exitosas de manera general, y han aportado un **feedback importante** que puede ayudar de manera muy positiva al futuro de AgroFind, pero antes de sacar conclusiones es importante denotar que la aplicación se ha probado con solo 5 usuarios y por tanto sería recomendable probarla con un número más amplio. Como nota de la aplicación en su totalidad, se queda en un **8.2 de media**, donde la **mínima es un 7** y la **máxima un 9**.

Lo más destacable ha sido la utilidad que se ha visto a AgroFind, en la que los dos agricultores se les ha preguntado sobre que es lo qué más les ha gustado y en que áreas mejorarían la aplicación.

> El diseño es bastante moderno y limpio, me recuerda a alguna aplicación que he probado anteriormente pero más actualizada, por haber usado el sistema PAC. Creo que mejoraría si tuviese algo para añadir nuevas parcelas desde la propia lista de eventos o poder hacer un cuaderno diario. [a]
>
> ---
>
> Lo que más me ha gustado es la facilidad de poder ver las parcelas, todas sus lindes y lo moderna que es. Lo que haría sería poner un paso a paso o manual, porque ya con mi edad estas cosas se me escapan un poco. [b]
>
> [a] **Iñaki T.**, 57 años. Agricultor en activo en la zona de las Merindades (Burgos).
> [b] **Heliodoro R.**, 78 años. Agricultor jubilado.

Estas citas dan mucha información para mejorar a futuro. Primeramente, **Iñaki** menciona el **cuaderno diario** con lo que se refiere al *cuaderno de campo o de explotación*, que es uno de los documentos que los agricultores tienen que rellenar para poder recibir ayudas, justificar sus cosechas y después presentar todo ello ante la Comunidad Autónoma pertinente (Castilla y León en su caso). Esto cambia entre CCAA, pero se podría tener un dosier por cada una y proporcionarlo desde la propia aplicación [106]. También comenta que se podría añadir por ejemplo, un botón de **añadir parcela en la lista de eventos**, para hacer el proceso más sencillo. Esto redirigiría al menú de mis parcelas y haría zoom en la ubicación actual.

**Heliodoro** explica que le gustaría que hubiera un **tutorial paso a paso** para aprender a usar AgroFind, y es un feedback muy valioso ya que es una de las funcionalidades que se había pensado hacer en un principio pero que se descartó durante el desarrollo. No tendría mucha complejidad técnica realmente, y haría la app más sencilla de usar, pero requiere de bastante tiempo en codificar por las múltiples pestañas que tiene.

Sobre feedback de otros usuarios, uno cree que habría que **diferenciar mejor entre ambos menús de parcelas**, y otro opina que un **botón para poder ver la contraseña** (que por defecto está oculta) en el inicio de sesión y en el registro sería útil. El último piensa que es una aplicación muy completa y no se le ocurren mejoras a futuro.

Todas estas ideas se canalizarán en el capítulo de conclusiones (véase capítulo 8) en la sección de trabajo futuro, junto a otros conceptos que el autor cree convenientes para AgroFind.

[106] Ejemplo del cuaderno agrícola digital de Junta de Castilla y León: https://agriculturaganaderia.jcyl.es/web/es/cuaderno-digital-explotacion-agricola.html

# 8. Conclusiones

Una vez finalizada la realización del trabajo, hay que comprobar qué resultados se han obtenido, si se han cumplido los objetivos propuestos, si ha habido alguna desviación en la planificación y qué se puede trabajar de cara a futuro para mejorar la aplicación. Además, se incluirán unas conclusiones personales del autor para dar cierre a esta memoria.

## 8.1. Conclusiones del trabajo

### 8.1.1. Cumplimiento de objetivos

Respecto a los objetivos que se plantearon desde un inicio, se hará una lista en la que se analiza cómo se han cumplido, y en caso de no haberse cumplido o haberse realizado parcialmente, las razones de ello.

- *Aplicación que permita a los agricultores llevar un control digital.*
  Se ha realizado una app que permite a los usuarios tener sus parcelas y sus actividades bajo control, y por tanto este objetivo se marca como completado satisfactoriamente.
- *Hacer más accesible la digitalización del campo.*
  AgroFind intenta ser lo más accesible posible, pero hay que mencionar que en las pruebas que se han realizado con los agricultores, se han encontrado puntos de mejora. Por tanto, este objetivo queda completado parcialmente, pero que con mejoras sencillas puede darse por satisfactorio.
- *Ofrecer una trazabilidad completa de la cosecha.*
  Al usar la blockchain con el contrato inteligente personalizado, se realiza toda la trazabilidad, pero en el informe al que se accede mediante el código QR solo se muestra esto en la cosecha, por tanto, se ha marcado como objetivo completado parcialmente.
- *Reducir la necesidad de infraestructura compleja.*
  La aplicación es completamente *serverless*, y por tanto se evita toda la complejidad de mantener un servidor. Objetivo cumplido satisfactoriamente.
- *Fomentar el uso de tecnologías digitales en el sector agrario.*
  AgroFind intenta acercar el uso de aplicaciones más modernas a agricultores, por tanto, se da como objetivo cumplido satisfactoriamente.
- *Desarrollar una prueba de concepto funcional.*
  Se ha creado una aplicación como prototipo, en la que se han introducido todo tipo de elementos que han llevado a que este objetivo sea cumplido satisfactoriamente.

### 8.1.2. Desviación de la planificación

Esta sección contiene la tabla 8.1, con las horas estimadas y reales de cada tarea.

| **Sección/Tarea** | **Estimado** | **Real** |
|---|---|---|
| 1.1. Elección de tema | 12 h | 28 h |
| 1.2. Reuniones de seguimiento | 12 h | 12 h |
| 1.3. Definición de objetivos y alcance | 6 h | 8 h |
| 1.4. Definición de tareas | 8 h | 8 h |
| 1.5. Estimación de tiempos | 4 h | 2 h |
| 1.6. Evaluación de riesgos | 4 h | 4 h |
| **1. Planificación** | **46 h** | **62 h** |
| 2.1. Redacción del DOP | 16 h | 12 h |
| 2.2. Estructuración de la memoria | 4 h | 4 h |
| 2.3. Redacción de la memoria | 80 h | 120 h |
| 2.4. Preparación de la defensa | 64 h | 46 h |
| **2. Documentación** | **164 h** | **182 h** |
| 3.1.1. Lectura de trabajos previos | 16 h | 16 h |
| 3.1.2. Revisión de antecedentes | 16 h | 12 h |
| 3.1.3. Búsqueda de fuentes de datos | 16 h | 20 h |
| 3.2.1. Aprendizaje de Dart y Flutter | 32 h | 32 h |
| 3.2.2. Aprendizaje de smart contracts | 12 h | 4 h |
| 3.2.3. Revisión de conocimientos AWS | 8 h | 16 h |
| 3.2.4. Aprendizaje de uso de SIGPAC | 16 h | 16 h |
| 3.2.5. Aprendizaje de uso de datos geográficos | 12 h | 20 h |
| **3. Recopilación de información** | **128 h** | **136 h** |
| 4.1. Análisis de la lógica de negocio | 8 h | 12 h |
| 4.2. Planteamiento de la arquitectura | 16 h | 16 h |
| 4.3. Diseño de la lógica de negocio | 24 h | 32 h |
| 4.4. Diseño de las interfaces | 18 h | 16 h |
| **4. Análisis y diseño** | **66 h** | **76 h** |
| 5.1. Desarrollo de la autenticación | 32 h | 40 h |
| 5.2. Desarrollo de la lógica de negocio | 72 h | 80 h |
| 5.3. Creación del smart contract | 16 h | 8 h |
| 5.4. Despliegue del smart contract | 12 h | 4 h |
| 5.5. Desarrollo de las interfaces | 36 h | 32 h |
| 5.6. Implementación de la arquitectura en AWS | 44 h | 56 h |
| **5. Desarrollo e implementación** | **212 h** | **220 h** |
| 6.1. Pruebas técnicas | 24 h | 40 h |
| 6.2. Pruebas de usuario | 24 h | 16 h |
| **6. Pruebas** | **48 h** | **56 h** |
| **Total** | **664 h** | **732 h** |

**Tabla 8.1:** *Horas estimadas y reales por cada sección y tarea del EDT.*

Como se ha visto en la anterior tabla, ha habido un incremento de **68 horas** frente a la planificación, y esto conlleva que haya que volver a calcular los costes totales del proyecto, en la siguiente sección.

### Costes totales

El coste final total del proyecto se desglosa y calcula al igual que en el estimado, pero utilizando las horas reales en la tabla 8.2, donde CTE es el *coste total estimado* calculado en la sección 4.4.

| **Concepto** | **Importe (€)** |
|---|---|
| Costes directos - Personal | $732\text{h} \cdot 12{,}67€/\text{h} + 28\text{h} \cdot 21{,}65€/\text{h}$ **9880.64 €** |
| Costes directos - Hardware | **213,19 €** |
| **Subtotal costes directos** ($SCD$) | **10.093,83 €** |
| Gastos indirectos (3 %, $GI$) | $SCD \cdot 0{,}03 = 302{,}81$ € |
| **Coste total final** (CTF) | $GI + SCD =$ **10.396,64 €** |
| **Desviación** | 10.396,64 (CTF) − 9.507,79 (CTE) = **888,85€** |

**Tabla 8.2:** *Resumen del coste final del proyecto y desviación respecto al coste estimado.*

La causa principal de esta desviación es el haber **estimado incorrectamente los tiempos de las tareas**. Por un lado, las relacionadas con la blockchain y el smart contract se han sobreestimado, porque finalmente el contrato ha sido relativamente sencillo y el conocimiento previo sobre el funcionamiento de validarlo en Polygon ha ayudado mucho. Por el otro, la redacción de memoria o las pruebas técnicas se han subestimado, la primera por la extensión de la misma, ya que al haber realizado tanto trabajo, plasmarlo en detalle es extenso. Y la segunda, porque intentar probar para tantos tipos de dispositivos distintos e intentar cubrir el mayor número de líneas de código con *tests* lleva mucho más trabajo de lo que se pensaba en un principio.

## 8.2. Conclusiones personales

AgroFind ha sido el culmen de todos los conocimientos adquiridos tanto en el grado, como en las prácticas laborales como fuera de la universidad.

El aplicar todo lo aprendido a un proyecto tan extenso ha sido muy gratificante, ya que es algo que siento como propio y más que hacer un TFG creo que he realizado una prueba de concepto de algo utilizable en la vida real.

Ha habido momentos complicados, porque este proyecto ha sido desarrollado durante muchas horas de trabajo, tanto para las fases iniciales de aprendizaje de Flutter, profundizaje en AWS, la decisión de usar SIGPAC como herramienta para la gestión de las parcelas (con toda la complejidad que esto añadió al proyecto) como para las fases más de pruebas y documentación. Esto ha llevado a instantes cercanos al *burnout*, que se han mitigado con la motivación de completar el TFG.

Sobre las conclusiones más directas en relación con áreas específicas, destacaría varias de ellas:

- **La facilidad de uso es muy importante**:
  Diseñar una interfaz sencilla e intuitiva es mucho más complejo de lo que parece a priori, especialmente cuando el público objetivo son agricultores con conocimiento limitado de la tecnología. Este proyecto me ha hecho pensar que la adopción real de soluciones digitales en el ámbito agrícola depende en gran medida de eliminar barreras en la experiencia de usuario, y no tanto en reinventar la rueda para el campo.
- **Equilibrar simplicidad y funcionalidad es un reto**:
  Durante el desarrollo, me he dado cuenta que ofrecer muchas funciones puede comprometer la facilidad de uso. Elegir qué incluir y qué no es una decisión crítica para evitar saturar la interfaz de la aplicación.
- **Utilizar SIGPAC es complejo pero merece la pena**:
  Incorporar el sistema creado por el MAPA ha supuesto un esfuerzo técnico notable, pero su valor añadido en términos de precisión y legitimidad en la identificación de parcelas justifica la dificultad (véase 8.3).
- **El uso de blockchain requiere explicación para el usuario**:
  Una tecnología como esta no aporta valor si el usuario final no la comprende. Este proyecto me ha llevado a darle importancia a la necesidad de acompañar la tecnología con una comunicación clara, explicándola de forma comprensible.
- **Trabajar con servicios cloud exige aprender más allá del desarrollo**:
  El utilizar servicios como AWS ha supuesto otro tipo de complicaciones por la necesidad de configurar correctamente permisos y despliegue, lo cual va más allá del código en sí. Hacer una aplicación *serverless* no implica necesariamente menor complejidad (incluso a veces es mayor), sino una distribución distinta del esfuerzo.

- **El desarrollo en solitario implica aprender a priorizar**:
  Trabajar completamente solo en un proyecto de esta magnitud obliga a tomar decisiones constantemente sobre qué es viable implementar dentro del tiempo del que se dispone, y aceptar que no todo lo planteado inicialmente puede materializarse sin comprometer la calidad del producto final.
- **El *feedback* de los usuarios es indispensable**:
  Recibir opiniones y sugerencias de agricultores reales ha permitido detectar fallos o posibles mejoras no previstas en un inicio, haciendo a su vez que el trabajo a futuro esté mucho más definido.
- **Crear proyectos con impacto real es motivador**:
  Desarrollar esta aplicación no solo me ha enseñado aspectos técnicos, sino que ha reforzado mi motivación por trabajar en proyectos con propósito y utilidad concreta. Esto contrasta un poco con los trabajos que se han hecho a lo largo de la carrera, ya que al trabajar tanto para uno como este, la finalización del mismo es muy satisfactoria.

Finalmente, tengo que agradecer a mi tutor del trabajo, puesto que me ha guiado en las decisiones más complicadas y ha hecho que sea mucho más agradable acabar con el proyecto de la manera que yo quería, sin sacrificar funcionalidades clave pero haciendo pequeños ajustes sobre el alcance del mismo para poder realizarlo todo. También creo que tengo que darme crédito a mí mismo, debido a que he dedicado mucho esfuerzo y tiempo en que AgroFind tome forma, por convicción propia en intentar ayudar al campo.

## 8.3. Trabajo futuro

La última sección de la memoria explica que mejoras se pueden hacer en AgroFind de cara a futuro, cómo se podrían enfocar las mismas y la necesidad de alguna de ellas para pasar de una prueba de concepto a un producto utilizable y con una posible puesta en marcha.

A continuación se mencionan ideas que mejorarían la aplicación, tanto elementos que el autor no ha llegado a desarrollar del todo, cómo conceptos que los usuarios han sugerido como *feedback* durante las pruebas.

#### Integración con DNI Electrónico para verificación de identidad

Se propone incorporar un sistema de autenticación que utilice el DNI Electrónico para la identidad del usuario que registra o gestiona una determinada parcela.

Por un lado, se debería contemplar la posibilidad de declarar distintas relaciones con la parcela (propietario, arrendatario, gestor autorizado...), ya que no todos los agricultores son necesariamente los propietarios legales de los terrenos que cultivan. Por el otro, sería muy importante el cumplimiento de la LGPD y demás legislación vigente, como se ha analizado en la sección 6.1.4.

**Cambio de base de datos para soporte multiplataforma completo.**

Para garantizar la compatibilidad total de la aplicación con los diferentes sistemas operativos y plataformas, sería necesario cambiar la base de datos local a una que ofrezca mayor flexibilidad y soporte multiplataforma como *drift* (véase 7.2.3).

**Visualización de la trazabilidad completa.**

Aunque la información relacionada a toda la trazabilidad está visible en la blockchain y almacenada en DynamoDB, el informe que se muestra en la aplicación solo contiene información de la última cosecha. Podría ser de mayor utilidad tener un informe con toda la información al completo, pese a estar de manera pública. Para ello sería necesario hacer cambios en la infraestructura de AWS al generar estos archivos.

**Incorporación de un tutorial interactivo en la primera ejecución.**

Para mejorar la experiencia de los usuarios, especialmente entre agricultores con escasa familiaridad con herramientas digitales, podría ser de mucha ayuda incluir un tutorial interactivo (cómo explicó Heliodoro en la sección 7.3.2) que se muestre automáticamente la primera vez que se accede a la aplicación.

Aunque ya se ha desarrollado la lógica para detectar el primer inicio mediante el uso de *SharedPreferences*, faltaría diseñar y programar el contenido didáctico de dicho tutorial.

**Añadir eventos o parcelas de manera más directa.**

Para simplificar la introducción de nuevos datos en la aplicación, se podría añadir un acceso directo para la creación de parcelas o eventos desde el propio menú de eventos. Esta funcionalidad es algo que los propios labradores han solicitado en las pruebas de usuario, ya que podría mejorar la accesibilidad y la eficiencia en el uso diario de la aplicación.

**Ampliar pruebas a iOS y macOS.**

Aunque ya se han realizado pruebas básicas de compilación en iOS utilizando Codemagic, todavía no se ha llevado a cabo una validación completa de la aplicación en estos sistemas. Es necesario ejecutar pruebas funcionales y de interfaz en dispositivos reales, así como verificar el correcto funcionamiento en diferentes versiones de iOS y macOS.

Además, sería recomendable ampliar la batería de pruebas automáticas (*unit tests*, *widget tests* e *integration tests*) para asegurar el comportamiento esperado en todas las plataformas.

Finalmente, se ha incluido una pequeña tabla con las prioridades a seguir en caso de realizar estas mejoras a futuro en AgroFind.

| **Mejora** | **Prioridad** |
|---|---|
| Integración con DNI Electrónico | **Baja** |
| Cambio a base de datos multiplataforma | **Alta** |
| Visualización completa de trazabilidad mediante QR | **Alta** |
| Incorporación de tutorial interactivo | **Media** |
| Hacer más directa la creación de eventos/parcelas | **Baja** |
| Ampliar pruebas a iOS y macOS | **Media** |

**Tabla 8.3:** *Tabla de prioridades de las propuestas de mejora.*

# Bibliografía


[1] Wolfert, S., Ge, L., Verdouw, C. y Bogaardt, MJ. Big data in smart farming | A review. Agricultural Systems, 153, 69–80, 2017.

[2] Ministerio de Agricultura, Pesca y Alimentación (MAPA) (2024) La transformación digital en la agricultura Española. Disponible en: https://www.mapa.gob.es/es/desarrollo-rural/temas/innovacion-medio-rural/dtt-6-transformacion-digital-agricultura-espanola-ministerio_tcm30-699542.pdf (Consultado: el 10 de junio de 2025).

[3] UPV/EHU (2025) Reglamento de Convivencia. Disponible en: https://www.ehu.eus/es/web/gardentasun-ataria/universidad-del-pa%C3%ADs-vasco/euskal-herriko-unibertsitateko-elkarbizitza-araudia (Consultado: el 11 de junio de 2025).

[4] UPV/EHU (2025) EHUagenda 2030 por el desarrollo sostenible. Disponible en: https://www.ehu.eus/es/web/iraunkortasuna/ehuagenda-2030 (Consultado: el 11 de junio de 2025).

[5] Financial Food (2022) Carrefour implanta la tecnología blockchain en sus productos bio. Disponible en: https://financialfood.es/carrefour-implanta-la-tecnologia-blockchain-con-sus-productos-bio/ (Consultado: el 23 de diciembre de 2024).

[6] Ministerio de Industria, Comercio y Turismo (2022) Balanza Comercial Agroalimentaria 2022. Disponible en: https://comercio.gob.es/ImportacionExportacion/Informes_Estadisticas/Historico_Balanza/Balanza_Comercial_Agroalimentaria_2022.pdf (Consultado: el 11 de junio de 2025).

[7] Ayuntamiento de Medina de Pomar (2025) Información sobre Moneo. Disponible en: https://www.medinadepomar.net/node/100 (Consultado: el 11 de junio de 2025).

[8] Statista (2024) Distribución del PIB de España por sectores económicos. Disponible en: https://es.statista.com/estadisticas/501643/distribucion-del-producto-interior-bruto-pib-de-espana-por-sectores\protect\penalty\z@-economicos/ (Consultado: el 19 de noviembre de 2024).

[9] Banco Mundial (sin fecha) Agricultura, silvicultura y pesca, valor agregado ( % del PIB). Datos desde 1960 hasta 2023. Disponible en: https://datos.bancomundial.org/indicador/NV.AGR.TOTL.ZS?end=2023&start=1960&view=map (Consultado: el 21 de noviembre de 2024).

[10] Parlamento Europeo (2021) Estadísticas sobre el empleo y la producción en la UE. Disponible en: https://www.

europarl.europa.eu/topics/es/article/20211118STO17609/estadisticas-sobre-la-agricultura-de-la-ue-ayudas-empleo-produccion (Consultado: el 21 de noviembre de 2024).

[11] Instituto Nacional de Estadística (2023) Encuesta sobre las explotaciones agrícolas. Disponible en: https://www.ine.es/dyngs/INEbase/operacion.htm?c=Estadistica_C&cid=1254736176854&menu=ultiDatos&idp=1254735727106 (Consultado: el 21 de noviembre de 2024).

[12] MAPA (2020) Una visión global de la Agricultura Española a través del análisis del censo agrario. Disponible en: https://www.mapa.gob.es/es/ministerio/servicios/analisis-y-prospectiva/informemapa_ca2020_tcm30-653742.pdf (Consultado: el 19 de noviembre de 2024).

[13] Ebro Valley Agro Tech (2021-2024) Disponible en: https://ebroagrotech.com/ (Consultado: el 23 de noviembre de 2024).

[14] Odin Solutions (sin fecha) Disponible en: https://www.odins.es/smart-agro/ (Consultado: el 23 de noviembre de 2024).

[15] MAPA (2023) Barómetro del clima de confianza del sector agroalimentario. Disponible en: https://www.mapa.gob.es/es/alimentacion/temas/consumo-tendencias/barometro-del-clima-de-confianza-del-sector-agroalimentario/ (Consultado: el 25 de noviembre de 2024).

[16] EIT Food (2023) EIT Food Trust Report. Disponible en: https://www.eitfood.eu/reports/trust-report-2023 (Consultado: el 25 de noviembre de 2024).

[17] Consejo de la Unión Europea (2003) Reglamento (CE) nº 1782/2003 del Consejo, de 29 de septiembre de 2003. Disponible en: https://eur-lex.europa.eu/legal-content/ES/TXT/?uri=CELEX%3A32003R1782 (Consultado: el 30 de noviembre de 2024).

[18] Parlamento Europeo y Consejo de la UE (2025) Reglamento que reemplaza al 1782/2023. Disponible en: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A02021R2115-20240525 (Consultado: el 30 de noviembre de 2024).

[19] Fondo Español de Garantía Agraria (FEGA) (sin fecha) Disponible en: https://sigpac.mapama.gob.es/fega/visor/ (Consultado: el 30 de noviembre de 2024).

[20] Diario Oficial de la Unión Europea (2007) Directiva 2007/2/CE del Parlamento Europeo. Disponible en: https://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2007:108:0001:0014:ES:PDF (Consultado: el 30 de noviembre de 2024).

[21] FEGA (2023) Detalles del servicio WMS de SIGPAC. Disponible en: https://www.fega.gob.es/es/pepac-2023-2027/sistemas-gestion-y-control/sigpac/wms-de-sigpac (Consultado: el 5 de diciembre de 2024).

[22] FEGA (sin fecha) Directorio de servicios de SIGPAC. Disponible en: https://www.mapa.gob.es/es/cartografia-y-sig/ide/directorio_datos_servicios/agricultura/servicios-wms-sigpac/wms_sigpac.aspx (Consultado: el 5 de diciembre de 2024).

[23] FEGA y MAPA (sin fecha) Propiedades de un recinto por coordenadas. Disponible en: https://sigpac-hubcloud.es/html/csp/consultas/cooordenadas.html ((Consultado: el 5 de diciembre de 2024).

[24] Serra, P., Moré, G. y Pons, X. Consecuencias en la cartografía de cultivos mediterráneos de la combinación de datos ráster-vector: enriquecimiento de una base de datos SIGPAC a través de teledetección. Universidad Nacional de Educación a Distancia, 2008.

[25] Millán, C., Alfonso, A. y Hernández, C. El nuevo SIGPAC DEHESA, una herramienta fundamental en la aplicación de la nueva normativa de cerdo Ibérico. Editorial Agrícola Española, 2008.

[26] Sánchez, N., Rodríguez, M., Martínez, J. y Calera, A. SIGPAC y series multitemporales LANSAT 15 TM como estrategia híbrida de clasificación de usos de suelo para aplicaciones hidrológicas. Universidad de Sevilla, 2010.

[27] Martínez, A., de la Riva, J. y Losada, J. Propuesta de mejora en el protocolo de análisis del riesgo de inundación: inclusión de datos del SIGPAC y análisis de vegetación mediante LiDAR. Universidad de Zaragoza, 2021.

[28] Statcounter (2025) Cuota de mercado de Sistemas Operativos Móviles en España. Disponible en: https://gs.statcounter.com/os-market-share/mobile/spain (Consultado: el 13 de diciembre de 2024).

[29] Microsoft (2025) Comparación entre aplicaciones nativas, híbridas y multiplataforma. Disponible en: https://www.microsoft.com/en-us/power-platform/products/power-apps/topics/app-development/native-vs-cross-platform-apps (Consultado: el 3 de enero de 2025).

[30] Oracle (2015) The Java Virtual Machine Specification. Documentación oficial sobre la JVM de Java. Disponible en: https://docs.oracle.com/javase/specs/jvms/se8/html/index.html (Consultado: el 19 de enero de 2025).

[31] IBM (2021-2024) Java Virtual Machine Management. Disponible en: https://www.ibm.com/docs/en/b2b-integrator/6.1.1?topic=management-java-virtual-machine (Consultado: el 19 de enero de 2025).

[32] Flutter (sin fecha) Flutter SDK. Disponible en: https://github.com/flutter/flutter (Consultado: el 12 de mayo de 2025).

[33] React Native (sin fecha) React Native Framework. Disponible en: https://github.com/facebook/react-native (Consultado: el 12 de mayo de 2025).

[34] .NET MAUI (sin fecha) .NET Multi-platform App UI. Disponible en: https://github.com/dotnet/maui (Consultado: el 12 de mayo de 2025).

[35] Kotlin (sin fecha) Kotlin Programming Language. Disponible en: https://github.com/JetBrains/kotlin (Consultado: el 12 de mayo de 2025).

[36] Stack Overflow (2024) Stack Overflow Developer Survey. Disponible en: https://survey.stackoverflow.co/2024/technology (Consultado: el 12 de mayo de 2025).

[37] Google Trends (2025) Comparación de interés de búsqueda mundialmente: Flutter, React Native, Kotlin y .NET MAUI. Disponible en: https://trends.google.es/trends/explore?cat=5&date=2025-01-01%202025-12-31&q=%2Fg%2F11f03_rzbg,%2Fm%2F0_lcrx4,%2Fg%2F11h03gfxy9,%2Fg%2F11mqkqmrcr&hl=es (Consultado: el 12 de mayo de 2025).

[38] Amazon Web Services (2024) ¿Qué es Flutter? - Explicación sobre Flutter. Disponible en: https://aws.amazon.com/es/what-is/flutter/ (Consultado: el 9 de abril de 2025).

[39] Flutter (sin fecha) Flutter Architectural Overview. Docs oficiales. Disponible en: https://docs.flutter.dev/resources/architectural-overview (Consultado: el 7 de marzo de 2025).

[40] Baseflow (sin fecha) Geolocator: Biblioteca de geolocalización de Flutter. Disponible en: https://pub.dev/packages/geolocator (Consultado: el 7 de marzo de 2025).

[41] Flutter (sin fecha) Clase Widget. API Flutter Developers. Disponible en: https://api.flutter.dev/flutter/widgets/Widget-class.html (Consultado: el 7 de marzo de 2025).

[42] Google (2025) Página oficial de Dart. Disponible en: https://dart.dev/ (Consultado: el 3 de marzo de 2025).

[43] Gonzalez, C. (2025) FullStack Labs | An Introduction to Flutter's World. Disponible en: https://www.fullstack.com/labs/resources/blog/an-introduction-to-flutters-world (Consultado: el 7 de marzo de 2025).

[44] Karasavvas, T. (2022) StackOverflow Blog | Why Flutter is the most popular cross-platform mobile SDK. Disponible en: https://stackoverflow.blog/2022/02/21/why-flutter-is-the-most-popular-cross-platform-mobile-sdk/ (Consultado: el 12 de mayo de 2025).

[45] Statista (2023) Average weekly working hours of software developers worldwide. Disponible en: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/ (Consultado: el 12 de mayo de 2025).

[46] Kim, W. Cloud computing: Today and tomorrow. *Journal of Object Technology*, 8(1):65–72, 2009.

[47] Mell, P. y Grance, T. The NIST Definition of Cloud Computing. *National Institute of Standards and Technology*, Special Publication 800-145, 2009.

[48] Bigelow, S. (2025) TechTarget | The history of cloud computing explained. Disponible en: https://www.techtarget.com/whatis/feature/The-history-of-cloud-computing-explained (Consultado: el 19 de mayo de 2025).

[49] Bell, C. G. Time shared computers. *Carnegie Institute of Technology*, 1967.

[50] International Data Corporation (2024) Global public cloud spending to hit $805B in 2024. Disponible en: https://my.idc.com/getdoc.jsp?containerId=prUS52460024 (Consultado: el 19 de mayo de 2025).

[51] Agencia EFE (2025) Alibaba invertirá 52.400 millones de dólares en IA y cloud en los próximos 3 años. Disponible en: https://efe.com/economia/2025-02-24/china-alibaba/ (Consultado: el 23 de mayo de 2025).

[52] Statista (2025) Amazon and Microsoft Stay Ahead in Global Cloud Market. Disponible en: https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/ (Consultado: el 24 de mayo de 2025).

[53] Amazon Web Services (sin fecha) Lista de servicios en la nube. Disponible en: https://aws.amazon.com/es/products/ (Consultado: el 24 de mayo de 2025).

[54] AWS (sin fecha) Lambda: Ejecución de código sin servidores. Disponible en: https://aws.amazon.com/es/lambda/ (Consultado: el 24 de mayo de 2025).

[55] AWS (sin fecha) S3: Simple Storage Service. Disponible en: https://aws.amazon.com/es/s3/ (Consultado: el 24 de mayo de 2025).

[56] AWS (sin fecha) DynamoDB: Base de datos NoSQL sin servidor. Disponible en: https://aws.amazon.com/es/dynamodb/ (Consultado: el 24 de mayo de 2025).

[57] AWS (sin fecha) API Gateway: Creación y gestión de API. Disponible en: https://aws.amazon.com/es/api-gateway/ (Consultado: el 24 de mayo de 2025).

[58] Netflix y AWS (sin fecha) Netflix Case Study. Disponible en: https://aws.amazon.com/es/solutions/case-studies/netflix-case-study/ (Consultado: el 24 de mayo de 2025).

[59] Kan, B., y Klein, D. AWS Corporate AI Use Cases. *Research Association for Interdisciplinary Studies* (RAIS), 151, 2022.

[60] AWS (sin fecha) Presupuestos de AWS, precios y servicios. Disponible en: https://aws.amazon.com/es/pricing/ (Consultado: el 24 de mayo de 2025).

[61] NIST (2009) The NIST Glossary | Blockchain. *National Institute of Standards and Technology*. Disponible en: https://csrc.nist.gov/glossary/term/blockchain (Consultado: el 24 de mayo de 2025).

[62] Johnson, D., Menezes, A. y Vanstone, S. The elliptic curve digital signature algorithm (ECDSA). *International Journal of Information Security*, 1, 36-63, 2001.

[63] Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008.

[64] Depoortère, C. Examining the Writings of Satoshi Nakamoto: A Monetary Analysis of the Bitcoin Protocol. *Review of Political Economy, 37(2)*, 392–411, 2024.

[65] Christidis, K. y Devetsikiotis, M. Blockchains and smart contracts for the internet of things. *IEEE access*, 4, 2292-2303, 2006.

[66] G. Ateniese, B. Magri, D. Venturi y E. Andrade. Redactable Blockchain – or – Rewriting History in Bitcoin and Friends. *IEEE European Symposium on Security and Privacy (EuroS&P)*, Paris, 2017.

[67] Lamport, L., Shostak, R., y Pease, M. The Byzantine generals problem. *In Concurrency: the works of Leslie Lamport*, pp. 203-226, 2019.

[68] Buterin, V. Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform, 2013.

[69] Ethereum (2025) What is Ethereum? Disponible en: https://ethereum.org/es/what-is-ethereum/ (Consultado: el 26 de mayo de 2025).

[70] Ethereum | Solidity (sin fecha) Documentación de Solidity. Disponible en: https://solidity-es.readthedocs.io/es/latest/ (Consultado: el 26 de mayo de 2025).

[71] Poon, J., y Dryja, T. The bitcoin lightning network: Scalable off-chain instant payments. 2016.

[72] Zubler, L. upsi - a decentralized STI tracing approach. Tesis doctoral en *Ostschweizer Fachhochschule* https://eprints.ost.ch/id/eprint/1217/, 2024.

[73] Kanani, J., Nailwal, S. y Arjun, A Matic Network whitepaper. 2013.

[74] Bjelic, M., Nailwal, S., Chaudhary, A. y Wenxuan D. POL: One token for all Polygon chains. 2021

[75] Polygon (2025) MATIC to POL Migration Is Now Live. Everything You Need to Know. Disponible en: https://polygon.technology/blog/matic-to-pol-migration-is-now-live-everything-you-need-to-know (Consultado: el 29 de mayo de 2025).

[76] Polygon (2025) The Beginner's Guide to Aggregated Blockchains. Disponible en: https://polygon.technology/blog/the-beginners-guide-to-aggregated-blockchains (Consultado: el 29 de mayo de 2025).

[77] Talent.com (2025) Salario medio de ingeniero informático junior. Disponible en: https://es.talent.com/salary?job=ingeniero+inform%C3%A1tico+junior (Consultado: el 30 de mayo de 2025).

[78] UPV/EHU (2025) Portal de transparencia | Retribuciones del personal. Disponible en: https://www.ehu.eus/es/web/gardentasun-ataria/ordainketak (Consultado: el 30 de mayo de 2025).

[79] Firebase (2025) Precios de Firebase. Disponible en: https://firebase.google.com/pricing?hl=es-419 (Consultado: el 1 de junio de 2025).

[80] WeatherAPI (2025) Precios de WeatherAPI. Disponible en: https://www.weatherapi.com/pricing.aspx (Consultado: el 1 de junio de 2025).

[81] AWS (2025) AWS Pricing Calculator. Disponible en: http://calculator.aws/#/ (Consultado: el 1 de junio de 2025).

[82] AWS Pricing Calculator (2025) Estimación de costes para 2000 usuarios. Disponible en: https://calculator.aws/#/estimate?id=36c28905a1f87340a33e0e37bd9c42b547ca03ce (Consultado: el 1 de junio de 2025).

[83] AWS Pricing Calculator (2025) Estimación de costes para 30000 usuarios. Disponible en: https://calculator.aws/#/estimate?id=24e1414824fc45622d7e306e873444c8acd22134 (Consultado: el 1 de junio de 2025).

[84] AWS (2025) Eligiendo la Clave de Partición Correcta en DynamoDB. Disponible en: https://aws.amazon.com/es/blogs/aws-spanish/choosing-the-right-dynamodb-partition-key/ (Consultado: el 1 de junio de 2025).

[85] Magic Labs (2025) Docs | Featured Chains | Polygon. Disponible en: https://magic.link/docs/blockchains/featured-chains/polygon (Consultado: el 2 de junio de 2025).

[86] Polygon Technology (2025) Meta-transactions – The current state of transacting. Disponible en: https://docs.polygon.technology/pos/concepts/transactions/meta-transactions/#the-current-state-of-transacting (Consultado: el 2 de junio de 2025).

[87] MetaMask Support (2025) ¿Qué son las metatransacciones? Disponible en: https://support.metamask.io/es/more-web3/learn/what-are-metatransactions-/ (Consultado: el 3 de junio de 2025).

[88] Geng, Y., Qin, B., Wang, Q., Shi, W. y Wu, Q. Subsidy Bridge: Rewarding Cross-Blockchain Relayers with Subsidy. In *International Conference on Information and Communications Security*, Springer, 2023, pp. 571–589.

[89] Starton (2025) Understanding the Relayer. Disponible en: https://docs.starton.com/docs/transactions/understanding-the-relayer (Consultado: el 3 de febrero de 2025).

[90] AWS (2025) Using Global Secondary Indexes in DynamoDB. Disponible en: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GSI.html (Consultado: el 10 de marzo de 2025).

[91] Android (2025) Cómo comenzar a usar Android Studio. Disponible en: https://developer.android.com/studio?hl=es-419 (Consultado: el 11 de febrero de 2025).

[92] Flutter (2025) Get Started in Flutter development. Disponible en: https://docs.flutter.dev/get-started/install/windows (Consultado: el 11 de febrero de 2025).

[93] Firebase (2025) Instalación de la CLI de Firebase. Disponible en: https://firebase.google.com/docs/cli?hl=es-419#setup_update_cli (Consultado: el 13 de febrero de 2025).

[94] GeeksforGeeks (2024) Package Diagram: Introduction, Elements, Use Cases and Benefits. Disponible en: https://www.geeksforgeeks.org/package-diagram-introduction-elements-use-cases-and-benefits/ (Consultado: el 4 de junio de 2025).

[95] Firebase (2025) Agregar Firebase a tu app de Flutter. Disponible en: https://firebase.google.com/docs/flutter/setup?hl=es-419&platform=android (Consultado: el 15 de febrero de 2025).

[96] SIGPAC Hubcloud (2025) Descripción del Servicio de Teselas Vectoriales de SIGPAC. Disponible en: https://sigpac-hubcloud.es/html/mvt/descServicio.html (Consultado: el 19 de febrero de 2025).

[97] StackExchange (2025) Geographic Information Systems | EPSG 3857 or 4326 for Web Mapping. Disponible en: https://gis.stackexchange.com/questions/48949/epsg-3857-or-4326-for-web-mapping (Consultado: el 20 de febrero de 2025).

[98] Proj.org (2025) Coordinate operations | Projections | Mercator. Disponible en: https://proj.org/en/stable/operations/projections/merc.html (Consultado: el 20 de febrero de 2025).

[99] IBM Think (2025) ¿Qué es la redundancia de datos? Disponible en: https://www.ibm.com/es-es/think/topics/data-redundancy (Consultado: el 21 de mayo de 2025).

[100] MAPA (2023) Superficies y producciones anuales de cultivos. Disponible en: https://www.mapa.gob.es/es/estadistica/temas/estadisticas-agrarias/agricultura/superficies-producciones-anuales-cultivos/ (Consultado: el 20 de febrero de 2025).

[101] FEGA y MAPA (2023) GetCapabilities del servicio WMS de SIGPAC. Disponible en: https://wms.mapa.gob.es/sigpac/wms?request=getcapabilities&Service=WMS (Consultado: el 19 de enero de 2025).

[102] IGN/CNIG (2025) Infraestructura de Datos Espaciales | Visualización de servicios. Disponible en: https://www.ign.es/web/ide-area-nodo-ide-ign (Consultado: el 19 de enero de 2025).

[103] Parlamento Europeo y Consejo de la Unión Europea (2014) Reglamento (UE) Nº 910/2014 del Parlamento Europeo y del Consejo de 23 de julio de 2014 sobre identificación electrónica y los servicios de confianza para las transacciones electrónicas en el mercado interior (Reglamento eIDAS). Disponible en: https://eur-lex.

europa.eu/legal-content/ES/TXT/?uri=CELEX%3A32014R0910 (Consultado: el 2 de junio de 2025).

[104] Parlamento Europeo y Consejo de la Unión Europea (2016) Reglamento (UE) 2016/679 del Parlamento Europeo y del Consejo, de 27 de abril de 2016, relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales y a la libre circulación de estos datos (RGPD). Disponible en: https://eur-lex.europa.eu/legal-content/ES/TXT/?uri=CELEX%3A32016R0679 (Consultado: el 2 de junio de 2025).

[105] Gobierno de España (2018) Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales (LOPDGDD). Disponible en: https://www.boe.es/eli/es/lo/2018/12/05/3 (Consultado: el 2 de junio de 2025).

[106] Gobierno de España (2022) Real Decreto 311/2022, de 3 de mayo, por el que se regula el Esquema Nacional de Seguridad (ENS). Disponible en: https://www.boe.es/eli/es/rd/2022/05/03/311 (Consultado: el 2 de junio de 2025).

[107] Gobierno de España (2020) Ley 6/2020, de 11 de noviembre, reguladora de determinados aspectos de los servicios electrónicos de confianza. Disponible en: https://www.boe.es/eli/es/l/2020/11/11/6 (Consultado: el 2 de junio de 2025).

[108] MIT (2002) Teller S. and Durand F. Ray Casting Lecture - *6.837 Computer Graphics (F02)*. Disponible en: https://groups.csail.mit.edu/graphics/classes/6.837/F02/lectures/6.837-10_rayCast.pdf (Consultado: el 3 de febrero de 2025).

[109] AWS (2025) Managing Lambda dependencies with layers. Disponible en: https://docs.aws.amazon.com/lambda/latest/dg/chapter-layers.html (Consultado: el 3 de diciembre de 2024).

[110] AWS (2025) Optimizing costs on DynamoDB tables. Disponible en: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-cost-optimization.html (Consultado: el 3 de diciembre de 2024).

[111] Elhemali, M., Gallagher, N., Gordon, N., Idziorek, J, et al. Amazon DynamoDB: A scalable, predictably performant, and fully managed NoSQL database service. *USENIX Annual Technical Conference 2022* (pp. 1037-1048), 2022.

[112] OKTA (2025) Principle of Least Privilege: Definition, Methods and Examples. Disponible en: https://www.okta.com/identity-101/minimum-access-policy/ (Consultado: el 2 de junio de 2025).

[113] Mena, A. (2025) Smart Contract desplegado en Polygon Amoy, RegistroEventosParcela. Disponible en: https://amoy.polygonscan.com/address/0x443a1fa4bf2a8b70529dda692518889cc45ff5d1#events (Consultado: el 11 de junio de 2025).

[114] Solidity Team (2025) Solidity 0.8.29 Release Announcement. 12 marzo 2025. Disponible en: https://soliditylang.org/blog/2025/03/12/solidity-0.8.29-release-announcement/ (Consultado: el 13 de marzo de 2025).

[115] Smart Bear (2024) What Is Unit Testing? Disponible en: https://smartbear.com/learn/automated-testing/what-is-unit-testing/ (Consultado: el 2 de junio de 2025).

[116] Runeson, P. A survey of unit testing practices. *IEEE software*, 23.4: 22-29, 2006.

[117] Flutter Cookbook (2025) Mock dependencies using Mockito. Disponible en: https://docs.flutter.dev/cookbook/testing/unit/mocking (Consultado: el 2 de junio de 2025).

[118] Youtube | AI With Flutter (2024) Unit Test with Mockito in Flutter. Disponible en: UnitTestwithMockitoinFlutter (Consultado: el 2 de junio de 2025).

[119] Firebase | Google (2025) Introducción a Firebase Test Lab. Disponible en: https://firebase.google.com/docs/test-lab/?hl=es-419&authuser=1#how_does_it_work (Consultado: el 4 de junio de 2025).

[120] Firebase | Google (2025) Run a Robo test (Android). Disponible en: https://firebase.google.com/docs/test-lab/android/robo-ux-test (Consultado: el 4 de junio de 2025).

[121] AWS (2025) How to test an API method in API Gateway. Disponible en: https://docs.aws.amazon.com/apigateway/latest/developerguide/how-to-test-method.html (Consultado: el 9 de junio de 2025).

[122] AWS (2025) Export an API from API Gateway. Disponible en: https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-export-api.html?icmpid=docs_apigateway_console (Consultado: el 9 de junio de 2025).

[123] YouTube (2025) API Gateway Tutorial. Disponible en: https://www.youtube.com/watch?v=XyzZYlS5fzo (Consultado: el 9 de junio de 2025).

[124] Postman (2025) Overview of Mock APIs. Disponible en: https://learning.postman.com/docs/design-apis/mock-apis/overview/ (Consultado: el 9 de junio de 2025).

[125] AWS (2025) Testing Lambda functions in the console. Disponible en: https://docs.aws.amazon.com/lambda/latest/dg/testing-functions.html (Consultado: el 23 de marzo de 2025).

[126] Codemagic Docs (2025) YAML Quickstart | Building a Flutter app. Disponible en: https://docs.codemagic.io/yaml-quick-start/building-a-flutter-app/ (Consultado: el 5 de junio de 2025).

[127] Tekartik (2024) Repositorio de SQFlite | Issues. Disponible en: https://github.com/tekartik/sqflite/issues/212 (Consultado: el 9 de junio de 2025).

[128] AWS Docs (2024) Enable CORS on a resource using the API Gateway console. Disponible en: https://docs.aws.amazon.com/apigateway/latest/developerguide/how-to-cors-console.html (Consultado: el 6 de junio de 2025).

[129] AWS Knowledge Center (2024) How do I troubleshoot CORS errors from my API Gateway API? Disponible en: https://repost.aws/knowledge-center/api-gateway-cors-errors (Consultado: el 6 de junio de 2025).

[130] Flutter Docs (2024) Platform integration | Desktop support for Flutter. Disponible en: https://docs.flutter.dev/platform-integration/desktop (Consultado: el 6 de junio de 2025).

[131] Leaflet Docs (2024) Layers | Tile Providers. Disponible en: https://docs.fleaflet.dev/layers/tile-layer/tile-providers (Consultado: el 30 de enero de 2025).

# Anexo 1. Encuesta de opinión

En este anexo se analizará la encuesta realizada para saber la opinión de los consumidores, a los que se explicó de esta manera:

> ¡Hola!
>
> Soy un estudiante de Ingeniería Informática y actualmente estoy desarrollando **AgroFind**, una aplicación destinada a agricultores para mi Trabajo Fin de Grado. Esta app les permitirá trazar sus productos agrícolas utilizando tecnología blockchain, con el objetivo de mejorar la transparencia en los alimentos que consumimos, ayudando a los productores con una aplicación fácil de usar y con ventajas para ellos.
> El propósito de esta encuesta es conocer la opinión de los consumidores sobre temas como la importancia de los productos locales, el conocimiento de nuevas tecnologías como blockchain, y el valor que se le da a la trazabilidad y la confianza en los alimentos que consumimos.
> La encuesta es completamente anónima, no recopila el correo electrónico y los datos recogidos se utilizarán únicamente con fines académicos para mi Trabajo de Fin de Grado. No te llevará más de 3 minutos completarla y me ayudas a conocer tu opinión para mejorar mi aplicación.
>
> ¡Muchas gracias por tu participación!

A continuación se irán exponiendo capturas de pantalla con las preguntas y los resultados de las mismas, sin tener en cuenta las preguntas abiertas, debido a que sería demasiado extenso incluir todas las respuestas.

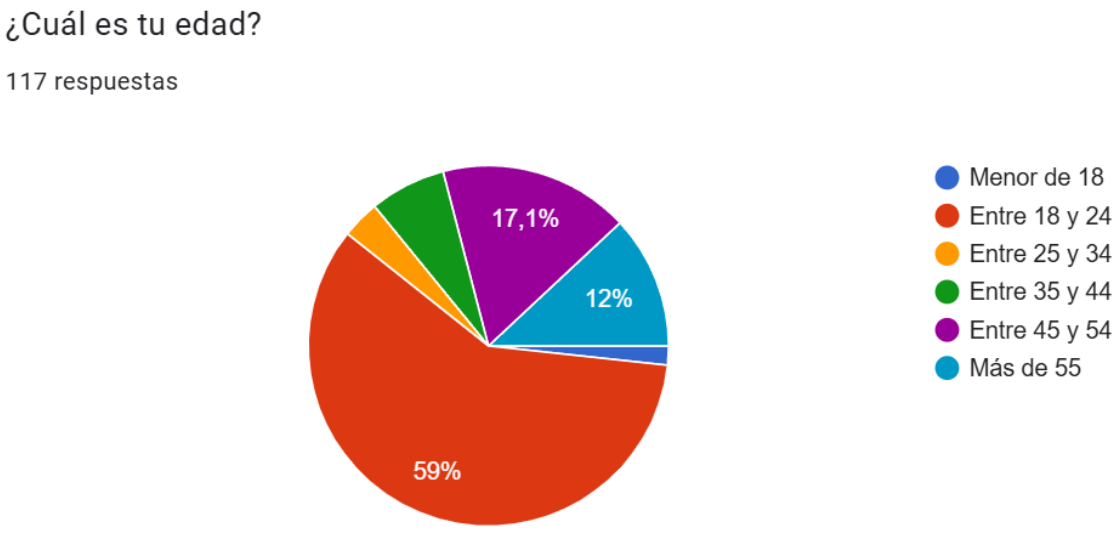



**Figura 8.1:** *¿Cuál es tu edad?*

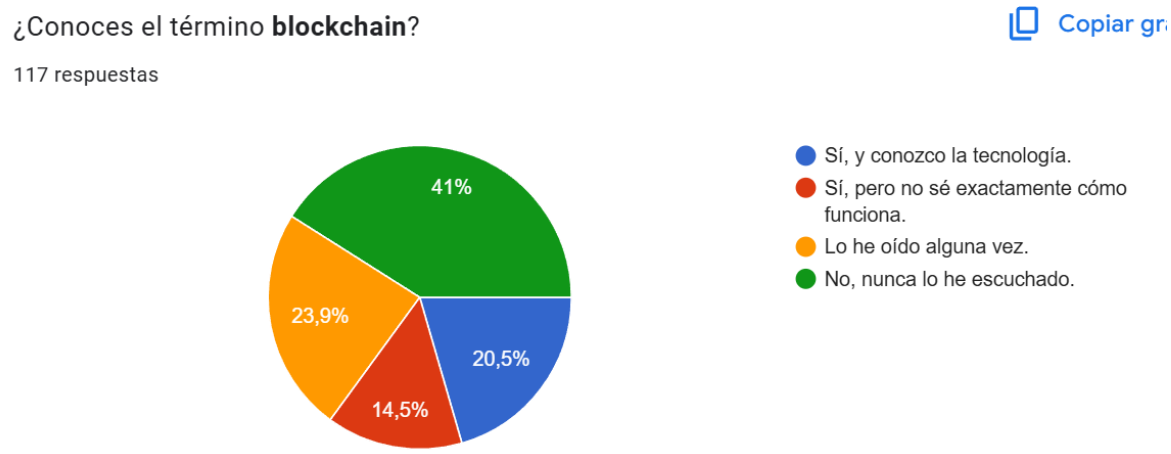



**Figura 8.2:** *¿Conoces el término blockchain?*

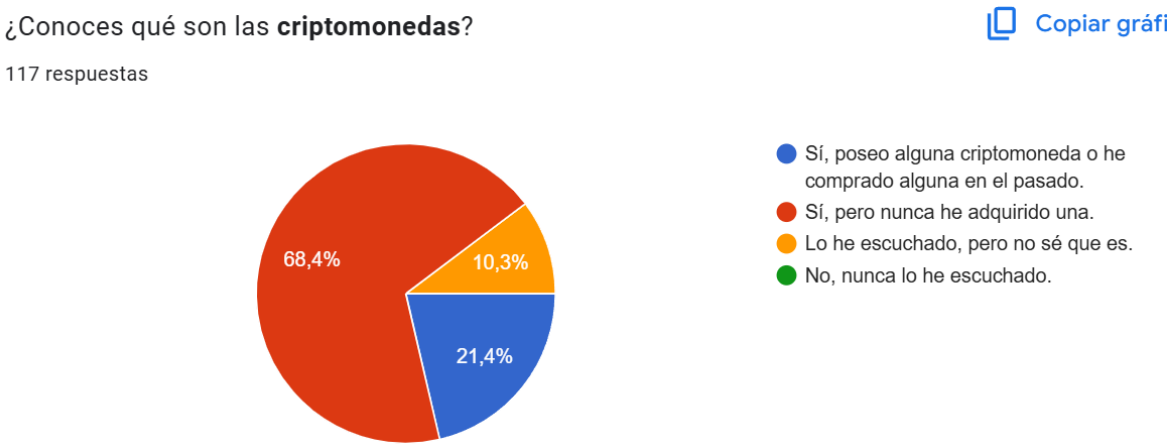



**Figura 8.3:** *¿Conoces qué son las criptomonedas?*

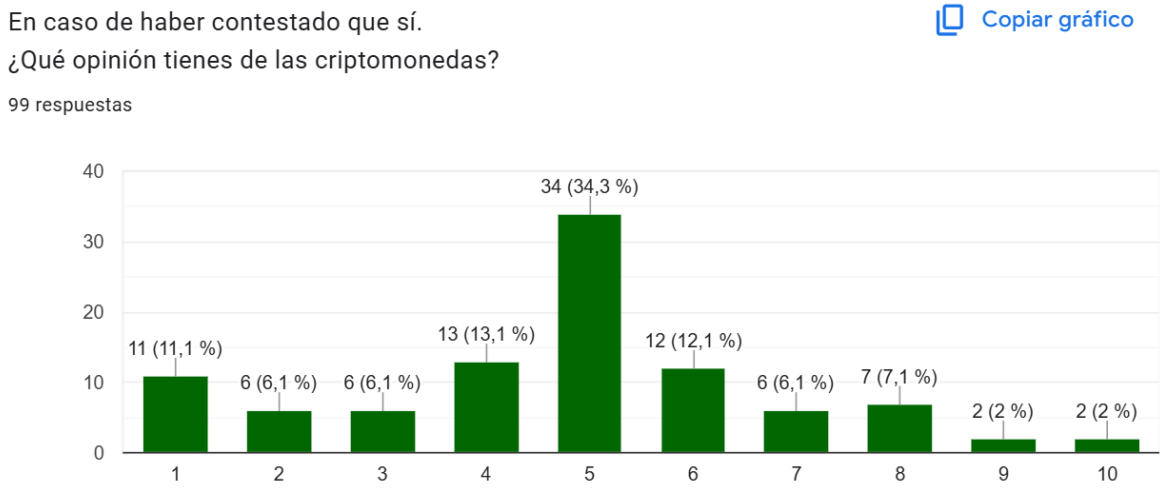



**Figura 8.4:** *En caso de haber contestado que sí, ¿Qué opinión tienes de las criptomonedas?*

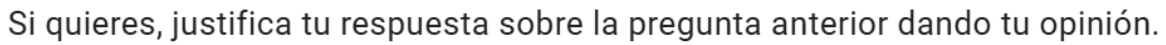



**Figura 8.5:** *Si quieres, justifica tu respuesta sobre la pregunta anterior dando tu opinión.*

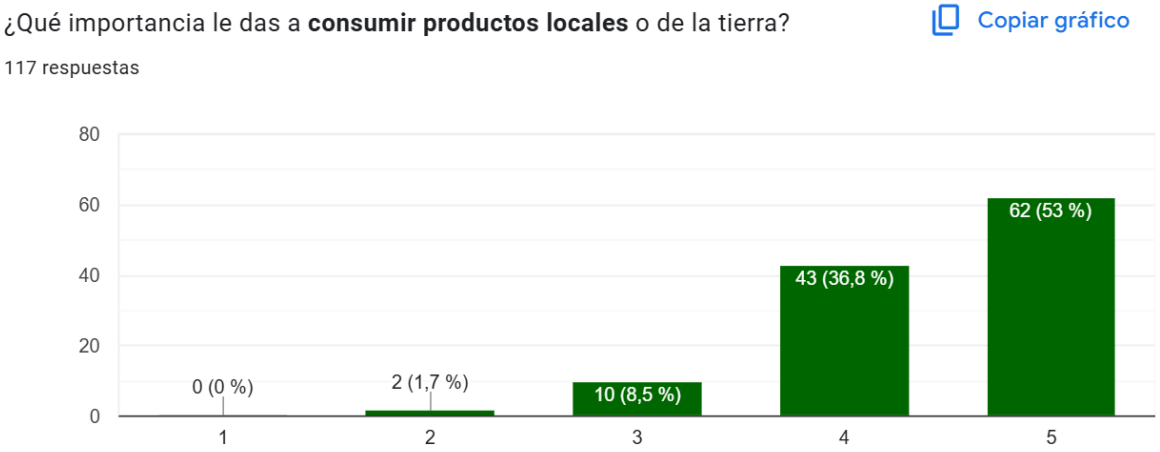



**Figura 8.6:** *¿Qué importancia le das a consumir productos locales o de la tierra?*

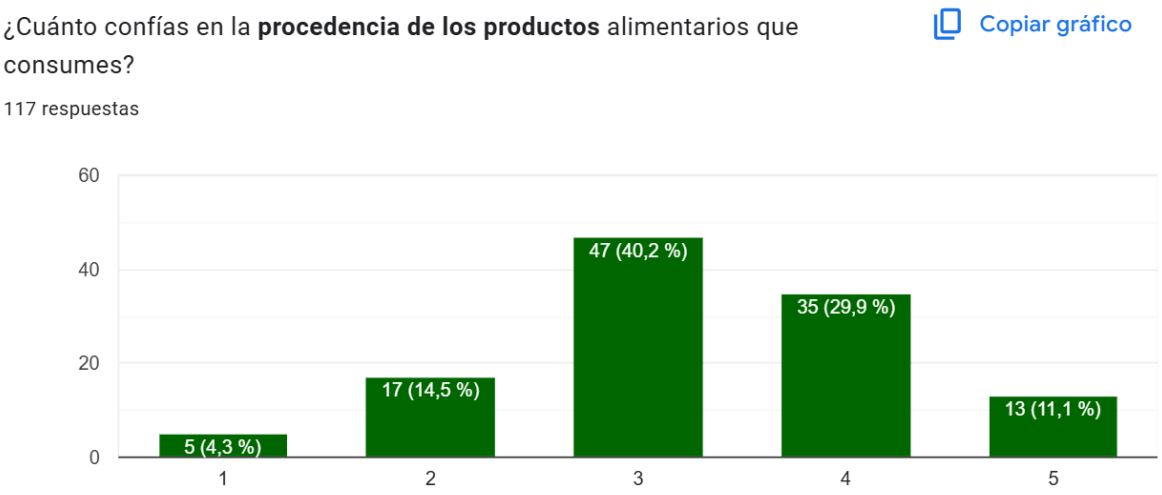



**Figura 8.7:** *¿Cuánto confías en la procedencia de los productos alimentarios que consumes?*

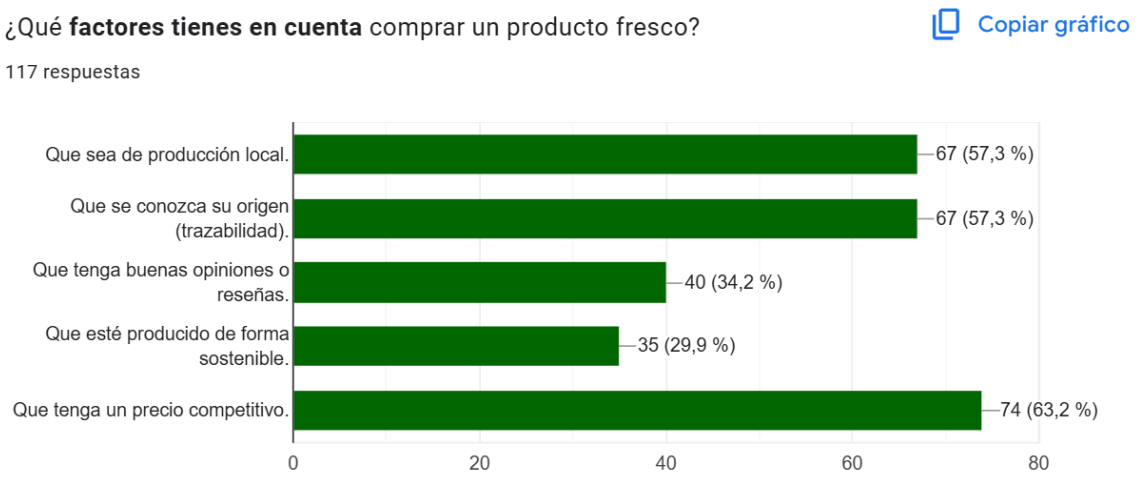



**Figura 8.8:** *¿Qué factores tienes en cuenta al comprar un producto fresco?*


**Figura 8.9:** *¿Cuánto estarías dispuesto/a a pagar por un producto local en comparación a uno extranjero?*

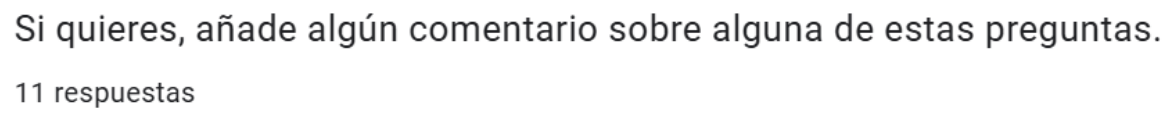



**Figura 8.10:** *Si quieres, añade algún comentario sobre alguna de estas preguntas.*