Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA (SG)
Grado en Ingeniería Informática de Servicios y Aplicaciones

Desarrollo full-stack de una aplicación web para la gestión de parcelas agrarias en React y NestJS

Alumno: David Pérez Hoyos

Tutores: Diego Martín de Andrés,
Francisco Hernando Gallego

Gracias a mi madre, a mi padre, a mi hermana, a mi familia en general, a mis amigos, a mi entorno, a la vida que me tocó vivir y a las infinitas casualidades que han hecho que yo esté escribiendo esto y tú lo estés leyendo.

Resumen

Cada temporada agrícola plantea retos en la gestión de parcelas: localizar límites exactos en mapas oficiales, obtener información oficial de SIGPAC y Catastro y evaluar el rendimiento económico de cada recinto. Esta plataforma web centraliza toda la información relevante: ubicación y delimitación de recintos, datos oficiales, imágenes satélite recortadas automáticamente y un modelo de IA que identifica hileras para calcular métricas como producción, ingresos y gastos.

Mediante análisis visuales interactivos, información económica personalizada y un acceso centralizado a los registros oficiales, se facilita la toma de decisiones y la planificación de campañas. La principal aportación es integrar en un único entorno web datos oficiales con análisis avanzados basados en imágenes satélite e inteligencia artificial. Para lograrlo, se emplean tecnologías de geoprocesamiento con Leaflet, componentes en React y Vite, autenticación segura mediante JWT y *refresh tokens*, y microservicios con NestJS y PostgreSQL que garantizan escalabilidad, coherencia de datos y facilidad de despliegue.

Palabras clave: aplicación web, agricultura de precisión, SIGPAC, catastro, satélite, inteligencia artificial, gestión de parcelas, análisis económico.

Abstract

Each agricultural season poses challenges in parcel management: locating exact boundaries on official maps, retrieving official data from SIGPAC and Cadastre, and assessing the economic performance of each plot. This web platform centralizes all relevant information: parcel location and delineation, official datasets, automatically cropped satellite imagery, and an AI model that detects vine rows to calculate metrics such as yield, revenue, and expenses.

Through interactive visual analysis, customized economic information, and centralized access to official records, it facilitates decision-making and campaign planning. The main contribution is to integrate official data with advanced analysis based on satellite imagery and artificial intelligence into a single web environment. To achieve this, it leverages geoprocessing with Leaflet, components built in React and Vite, secure authentication with JWT and *refresh tokens*, and a microservices architecture with NestJS and PostgreSQL to ensure scalability, data consistency, and ease of deployment.

Keywords: web application, precision agriculture, SIGPAC, cadastre, satellite imagery, artificial intelligence, parcel management, economic analysis.

Índice general

Índice de figuras

Índice de figuras

Índice de tablas

Índice de tablas

Capítulo 1

Introducción

Según datos oficiales, la edad media de los agricultores en España supera los 60 años. La sabiduría y cultura popular sugieren que las personas de mayor edad pueden tener dificultades para adaptarse a las nuevas tecnologías. A primera vista, esto podría llevar a pensar que desarrollar una aplicación de gestión agraria es un proyecto de escaso recorrido y aceptación. Sin embargo, la digitalización en el campo avanza a pasos agigantados: desde sensores IoT hasta plataformas de análisis satelital, el sector agrario está incorporando herramientas digitales que mejoran la eficiencia, reducen costes y aseguran prácticas más sostenibles.

Más allá de la brecha generacional, existe un enorme valor en facilitar el acceso a datos oficiales (SIGPAC, Catastro), imágenes por satélite y métricas económicas en un único panel de control. Al proporcionar una interfaz clara, intuitiva y adaptada a las necesidades reales de técnicos y gestores agrarios, se puede mitigar la resistencia inicial al cambio y demostrar los beneficios tangibles de la tecnología.

1.1. Motivación

En septiembre de 2024 tuve mi primer contacto con el mundo profesional como becario en CEPSA (ahora Moeve). Durante cuatro meses me desempeñé como un novizo desarrollador full-stack, y, aunque en ese periodo me di cuenta de que aquella no era mi pasión, puedo decir que aprendí bastante del oficio y le cogí cierto gusto. Al final, ser desarrollador full-stack se traduce en ser creador, en crear cosas. Cosas que puedes ver a medida que las creas. Que van tomando forma y que dependen solo de ti. Que nunca existirían de no ser por ti, o al menos no de la misma forma.

Es por ello que me decidí a hacer este proyecto; aunque en un primer momento no sería una aplicación de gestión agraria. Esto último se lo debo a mi co-tutor, Francisco, que me lo propuso como un proyecto empresarial real en el que no estaba yo solo: otros compañeros se encargarían de realizar otras funcionalidades, como el modelo de inteligencia artificial implementado. Yo sería el encargado de desarrollar el front y el back y "unir todas las piezas".

Así, de esta forma nació mi motivación personal: desarrollar una aplicación útil en equipo.

La motivación existencial del proyecto se encuentra unos párrafos más arriba.

1.2. Objetivos del proyecto

El objetivo último del proyecto es claro: desarrollar una plataforma web integral que facilite a técnicos y gestores agrarios la gestión, visualización y análisis de sus parcelas y recintos, combinando datos oficiales (SIGPAC y Catastro), imágenes satélite procesadas, datos económicos personalizados y un modelo de IA para optimizar la toma de decisiones agrícolas y económicas.

A continuación, se describen los objetivos específicos:

Tabla 1.1: Objetivos del proyecto

Código	Objetivo
O-1	**Autenticación y gestión de usuarios:** permitir registro, inicio de sesión y edición de perfil con autenticación basada en JWT y *refresh tokens*.
O-2	**Gestión de parcelas y recintos:** registrar múltiples parcelas por usuario, con referencia catastral y división automática de recintos según datos SIGPAC.
O-3	**Visualización geoespacial:** mostrar en un mapa interactivo (*Leaflet*) los límites oficiales de parcelas y recintos, con superposición de imágenes satélite.
O-4	**Procesado de imágenes satélite:** recortar automáticamente cada parcela a partir de la imagen satélite descargada (*Google Static Maps*) y generar máscaras a nivel de recinto.
O-5	**Análisis IA de viñedos:** utilizar un modelo entrenado para identificar hileras y extraer métricas asociadas.
O-6	**Introducción de datos económicos:** permitir al usuario indicar producción (kg), precios de venta/compra y otros costes para cada recinto.
O-7	**Visualización de métricas económicas:** calcular y mostrar ingresos, costes y beneficio neto.
O-8	**Integración de datos oficiales y meteorológicos:** sincronizar datos SIGPAC y ofrecer previsiones meteorológicas locales para apoyar la planificación de campañas.
O-9	**Interfaz responsive y accesible:** diseñar componentes en *React/Vite* con *Tailwind* para asegurar usabilidad en diferentes dispositivos y tamaños de pantalla.
O-10	**Escalabilidad y mantenimiento:** estructurar la aplicación en microservicios (*NestJS + PostgreSQL*) con buenas prácticas de código para facilitar futuras ampliaciones.

1.3. Alcance del proyecto

Los objetivos definidos anteriormente se engloban dentro de un marco a priori inamovible, el alcance del proyecto, el cual incluye:

- Desarrollo completo del frontend (React, Vite) backend (NestJS, TypeORM, PosgreSQL).
- Autenticación segura, CRUD de usuarios, parcelas y recintos.
- Visualización geoespacial y procesado de imagen satélite.
- Módulo IA para viñedos y análisis económico básico.
- Implementación de APIs: SIGPAC, Catastro, OpenWeather.

En concreto, la aplicación contará con las siguientes características principales:

Gestión de usuarios:

- **Registro, login y logout:** Los usuarios podrán registrarse en cualquier momento introduciendo su nombre, email, contraseña y localidad y municipio habituales. A su vez, se identificarán en la aplicación mediante el par correo electrónico y contraseña.
- **Modificar perfil:** Los usuarios podrán modificar cualquier dato de su perfil.
- **Eliminar perfil:** Los usuarios podrán eliminar su perfil junto con todos sus datos asociados (parcelas asignadas, datos económicos, etc).
- **Autenticación segura:** Implementación de JWT (*access + refresh tokens*) con protección de rutas en frontend y validación de tokens en backend. Refresco automático de sesión mediante cookies `HttpOnly`.

Gestión de parcelas:

- **Atributos identificadores:** Cada parcela contará con atributos identificadores para el Catastro (referencia catastral), y el SIGPAC (provincia, municipio, agregado, zona, polígono, parcela), así como otros atributos útiles.
- **Restricciones:** El usuario no podrá modificar ni los atributos ni la geometría oficial de las parcelas.
- **Alta, edición y baja de parcelas:** CRUD completo sobre entidades parcela.
- **Personalización:** El usuario podrá definir y modificar el nombre, descripción y notas de cada parcela asociada a su cuenta.
- **Búsqueda, filtrado y ordenación de parcelas:** Operaciones sobre la lista de parcelas del usuario.

Gestión de recintos:

- **Subdivisión:** Cada parcela se divide en uno o más recintos, cada uno con atributos específicos obtenidos gracias a SIGPAC.

- **Restricciones:** El usuario no podrá modificar ni los atributos ni la geometría oficial de los recintos.
- **Restricciones adicionales:** El usuario no podrá eliminar recintos de las parcelas. Recordar que cada recinto está asociado unívocamente a una parcela.
- **Datos económicos:** Cada recinto contará con datos económicos definidos por el usuario, como Variedad, Producción, Coste, Precio de Venta, etc.
- **Edición económica:** El usuario podrá modificar los datos económicos de cada recinto.
- **Obtención de datos SIGPAC:** Se cargarán las "traducciones" de diversos códigos SIGPAC al inicio de la aplicación para una correcta experiencia de usuario.

Visualización geoespacial:

- **Mapa interactivo:** El usuario podrá seleccionar y agregar a su cuenta cualquier parcela en territorio español a través de un mapa interactivo, mostrando la geometría de cada parcela y sus recintos.
- **Recorte automático de parcelas:** Generación automática de imágenes con fondo transparente para cada parcela y sus recintos.
- **Procesamiento mediante IA de viñedos:** Ejecución de un modelo de IA para detección de hileras, cómputo de longitudes y otros datos.

Información metereológica:

- **Consultas:** Consultas a servicio externo para mostrar estado y previsión climatológica de cada ubicación, tanto de las parcelas como la habitual del usuario.

Informes económicos:

- **Dashboards:** Creación de dashboards económicos con métricas que representen el estado económico del usuario.

Implementación:

- **Estructura modular:** El desarrollo se hará de forma modular, de modo que la aplicación sea fácilmente escalable en cuanto a funciones.
- **Seguridad y validación:** Sanitización y validación exhaustiva de todos los inputs, gestión de errores controlada, protección contra inyección de código y accesos no autorizados.
- **Configuración de entornos:** Distinción entre entornos de desarrollo y producción mediante variables de entorno para facilitar el despliegue en contenedores o servicios cloud.

Capítulo 2

Contexto y análisis del problema

El desarrollo de cualquier sistema software requiere partir de una comprensión profunda del entorno en el que se enmarca, así como de los retos y limitaciones que este plantea. Este capítulo tiene como objetivo analizar el contexto actual y justificar la necesidad de la solución propuesta. En primer lugar, se revisa el **estado del arte**, destacando las plataformas y herramientas más relevantes en gestión agraria, sus aportaciones y sus limitaciones. A continuación, se aborda el papel de la **inteligencia artificial** en el sector agrícola, con énfasis en las aplicaciones más comunes, sus beneficios y los desafíos que plantea. Finalmente, se presenta la **propuesta de solución**, que integra las lecciones aprendidas del análisis previo y define los pilares sobre los que se construirá la plataforma desarrollada en este trabajo.

2.1. Estado del arte

En los últimos años han surgido numerosas soluciones tecnológicas destinadas a la digitalización del sector agrario, un ámbito tradicionalmente rezagado en la adopción de herramientas informáticas. Estas iniciativas responden a la necesidad de optimizar la gestión de explotaciones agrícolas, mejorar la toma de decisiones y garantizar la sostenibilidad a largo plazo.

Plataformas de gestión agraria

Existen aplicaciones comerciales que permiten al agricultor llevar un control básico de sus fincas, cultivos y labores. Entre ellas destacan *Agroptima*, *Cropwise* o *Agropt*, que ofrecen funcionalidades como el registro de tratamientos fitosanitarios, control de gastos y generación de cuadernos de campo electrónicos. No obstante, en la mayoría de los casos, estas soluciones presentan dos limitaciones importantes: por un lado, requieren de suscripciones de pago que pueden suponer una barrera para explotaciones pequeñas; por otro, su integración con fuentes oficiales de información como SIGPAC o Catastro es reducida, obligando al usuario a introducir manualmente datos que ya se encuentran disponibles en registros públicos.

Herramientas cartográficas y visualización geoespacial

Por otra parte, organismos oficiales como el Ministerio de Agricultura y las comunidades autónomas ponen a disposición de los agricultores visores cartográficos en línea

basados en datos SIGPAC. Estos visores permiten consultar información oficial sobre recintos, superficies y usos de suelo. Sin embargo, se trata de herramientas de carácter consultivo, sin capacidad de personalización, análisis económico o almacenamiento de información adaptada a cada usuario. Del mismo modo, plataformas genéricas como *Google Earth* o visores cartográficos (GIS) de código abierto (por ejemplo, *QGIS*) ofrecen potentes funciones de visualización, pero requieren conocimientos técnicos avanzados y no están diseñadas para el uso cotidiano de agricultores o técnicos de campo.

Aplicaciones de teledetección e inteligencia artificial

En paralelo, la expansión de imágenes satelitales de libre acceso y la madurez de técnicas de visión por computador han dado lugar a proyectos que aplican inteligencia artificial para la detección de cultivos, estimación de índices de vegetación o predicción de rendimientos. Propuestas como *Cropin* o *EOSDA Crop Monitoring* han demostrado el valor de estas técnicas, aunque suelen orientarse a grandes explotaciones o empresas de servicios agronómicos, dejando de lado a pequeños y medianos productores. Además, su integración con datos económicos individuales y con registros oficiales sigue siendo limitada.

En conclusión, el estado del arte muestra un ecosistema fragmentado: por un lado, aplicaciones de gestión agraria con buenas capacidades administrativas pero escasa integración con datos oficiales y escasa personalización; por otro, visores cartográficos útiles pero poco prácticos en el día a día; y finalmente, soluciones de teledetección avanzadas, aunque dirigidas a usuarios especializados y a gran escala.

La propuesta de este proyecto se sitúa en un punto intermedio, buscando combinar en una única plataforma:

- La integración directa con fuentes oficiales (SIGPAC, Catastro).
- La automatización mediante imágenes satelitales y modelos de inteligencia artificial.
- El análisis económico adaptado a cada usuario.
- Una interfaz web sencilla y accesible, pensada para técnicos y agricultores sin necesidad de formación avanzada en informática.

De este modo, se aporta una solución integral que responde a necesidades reales del sector y que aprovecha las ventajas de la digitalización para democratizar el acceso a tecnologías avanzadas en el ámbito agrario.

2.2. Inteligencia artifical

El uso de técnicas de inteligencia artificial (IA) en el sector agrario ha crecido de manera exponencial en la última década, impulsado por la disponibilidad de grandes volúmenes de datos (imágenes satelitales, sensores IoT, registros históricos de producción) y por la necesidad de aumentar la eficiencia y sostenibilidad de las explotaciones. Estas técnicas permiten automatizar tareas que tradicionalmente han requerido de observación humana directa y, al mismo tiempo, proporcionan métricas cuantitativas que mejoran la toma de decisiones.

Aplicaciones actuales

Entre las aplicaciones más extendidas de la IA en agricultura se encuentran:

- **Clasificación y detección de cultivos:** mediante modelos de visión por computador entrenados con imágenes satelitales o de dron, es posible identificar el tipo de cultivo presente en una parcela, así como su estado de desarrollo.
- **Estimación de índices de vegetación:** técnicas de análisis de imágenes multiespectrales permiten calcular índices como NDVI o EVI, que actúan como indicadores de vigor y estrés del cultivo.
- **Predicción de rendimientos:** algoritmos de aprendizaje supervisado entrenados con series históricas de datos (clima, suelo, prácticas agrícolas) pueden predecir la productividad esperada de una campaña.
- **Detección de anomalías:** modelos de detección temprana permiten identificar problemas de plagas, enfermedades o estrés hídrico antes de que sean evidentes a simple vista.

Limitaciones y desafíos

A pesar de los avances, la aplicación práctica de la IA en agricultura presenta limitaciones relevantes:

- **Acceso desigual a datos y tecnología:** muchas soluciones están orientadas a grandes explotaciones o empresas con capacidad para invertir en sensores, drones y servicios de pago.
- **Generalización de modelos:** los algoritmos entrenados en determinadas regiones o cultivos no siempre son extrapolables a otros contextos, lo que limita su aplicabilidad universal.
- **Complejidad de uso:** en numerosos casos se requiere un conocimiento técnico avanzado para interpretar los resultados, lo que dificulta su adopción por agricultores de edad avanzada o con formación no tecnológica.

En definitiva, la inteligencia artificial se ha consolidado como una herramienta clave para transformar la agricultura, al ofrecer una visión más precisa y anticipada de los cultivos. Sin embargo, su impacto real depende de que estas tecnologías se adapten a las necesidades de los agricultores y gestores, combinando precisión técnica con accesibilidad y sencillez de uso. En este sentido, la presente aplicación busca situarse en un punto intermedio: aprovechar la potencia de modelos de IA especializados (en este caso, para viñedos) y presentarla en una interfaz intuitiva que acerque estos avances a un público amplio y heterogéneo dentro del sector agrario.

2.3. Propuesta de solución

La propuesta de este proyecto consiste en el desarrollo de una plataforma web integral para la gestión de parcelas y recintos agrícolas que unifique en un único espacio digital información oficial, datos satelitales, métricas económicas personalizadas y resultados de

modelos de inteligencia artificial. El objetivo es ofrecer una herramienta práctica, intuitiva y orientada al dato que apoye la planificación y toma de decisiones en el ámbito agrícola.

Siguiendo los objetivos propuestos en 1.1, la propuesta se diferencia de soluciones existentes al combinar en una sola plataforma las capacidades de gestión administrativa, la visualización geográfica, el análisis satelital y la dimensión económica, con un diseño centrado en la usabilidad. Se trata, por tanto, de una herramienta orientada a democratizar el acceso a tecnologías avanzadas, permitiendo que tanto técnicos como agricultores dispongan de información estratégica en un entorno único, accesible y fácil de utilizar.

Capítulo 3

Gestión y planificación del proyecto

La correcta gestión de un proyecto de software es tan importante como su implementación técnica. Una planificación adecuada permite distribuir recursos, coordinar tareas, definir plazos realistas y minimizar riesgos. En este capítulo se describen las tecnologías utilizadas, la metodología de desarrollo adoptada y la estimación de recursos necesarios para llevar a cabo el trabajo.

3.1. Tecnologías utilizadas

Herramientas de gestión

Para coordinar el desarrollo y facilitar el trabajo en equipo se emplearon distintas herramientas de apoyo:

- **Git y GitHub:** control de versiones y gestión colaborativa del código.
- **Trello:** tablero Kanban digital para registrar y priorizar tareas, organizadas en columnas (*Pendiente*, *En curso*, *Completado*).
- **Canva:** web de diseño gráfico utilizada para esbozos de interfaces de usuario y algunos diagramas.

En cuanto a la **estrategia de control de versiones** en GitHub, inicialmente se planteó un flujo de trabajo basado en la creación de ramas independientes para cada nueva *feature* o funcionalidad, con la posterior integración en la rama `main` mediante *merge requests*. Este enfoque hubiera permitido un mayor control de cambios y facilitado la revisión colaborativa.

Sin embargo, dado que el desarrollo fue llevado a cabo de forma individual, en la práctica se optó por un esquema más sencillo, trabajando directamente sobre la rama `main`. Esta decisión redujo la sobrecarga de gestión de ramas y resultó suficiente para un proyecto de estas características, aunque en un entorno profesional o con varios desarrolladores se recomienda mantener el flujo de ramas por funcionalidad para asegurar la calidad y trazabilidad del código.

Herramientas de desarrollo

Durante la implementación de la aplicación se utilizaron diversas tecnologías que facilitaron la escritura, ejecución y depuración del código:

- **Visual Studio Code:** editor principal de desarrollo, con extensiones para TypeScript, React y NestJS.
- **TypeScript:** lenguaje base del proyecto, que añade tipado estático sobre JavaScript, mejorando la robustez y mantenibilidad del código.
- **Node.js y npm:** entorno de ejecución y gestor de paquetes para instalar y ejecutar las dependencias tanto en frontend como en backend.
- **PostgreSQL:** sistema de gestión de bases de datos relacional utilizado para persistencia de datos.
- **DBeaver:** herramienta cliente para inspeccionar, consultar y gestionar la base de datos PostgreSQL de forma visual.
- **Python:** empleado para scripts externos de análisis de parcelas y generación de imágenes mediante IA.
- **MSW (Mock Service Worker):** librería para simular peticiones al backend durante el desarrollo del frontend.
- **Postman:** utilizado para probar y documentar manualmente los endpoints del backend, facilitando la depuración y validación de las APIs.

Frameworks y librerías

- **Frontend:** React con Vite como empaquetador, Redux Toolkit para gestión de estado, RTK Query para consumo de APIs y TailwindCSS para el diseño visual.
- **Backend:** NestJS como framework principal, TypeORM para el acceso a la base de datos y Passport-JWT para la autenticación.

3.2. Metodología de desarrollo

El proyecto se desarrolló siguiendo un enfoque **ágil y progresivo**. Los requisitos generales y las funcionalidades principales de la plataforma quedaron definidos desde el inicio, pero la forma concreta de implementarlos, así como ciertos aspectos de diseño y usabilidad, se fueron refinando a medida que avanzaba el desarrollo. Esto permitió adaptar la solución final a las recomendaciones del tutor y a las necesidades prácticas que surgían durante el trabajo.

La dinámica principal consistió en implementar de manera progresiva nuevas características —como el sistema de autenticación, la gestión de parcelas o la integración de imágenes satelitales— y presentarlas posteriormente al tutor. Estas revisiones periódicas funcionaban como instancias de validación, en las que se confirmaba lo ya construido, se detectaban posibles errores y se proponían ajustes o mejoras para el siguiente ciclo. De este modo, el proceso siguió un esquema de **iteración y retroalimentación**, muy alineado con los principios de las metodologías ágiles.

Para la organización de las tareas se recurrió a un tablero Kanban digital (Trello), en el que las actividades se clasificaban en tres columnas principales: *Pendiente*, *En curso* y *Completado*. Esta herramienta ofreció una visión global del estado del proyecto y ayudó a

priorizar qué funcionalidades debían abordarse primero en función de su relevancia y de la retroalimentación recibida.

Adicionalmente, se tomaron prácticas inspiradas en **Scrum**, como la entrega de resultados incrementales y la validación frecuente de los módulos desarrollados, lo que garantizó una evolución continua y una supervisión constante de la calidad. Aunque no se siguió de manera estricta un marco metodológico formal, este sistema de trabajo permitió mantener un desarrollo ágil, controlado y adaptativo.

En la práctica, la implementación siguió un orden incremental que permitió construir la aplicación desde sus cimientos hasta las funcionalidades más avanzadas. Se distinguen tres etapas, según el orden de implementación: desarrollo del frontend; desarrollo del backend; e integración final. En las tablas, se utiliza como referencia una **jornada de 5 horas** con objetivo de estimar el trabajo invertido en el proyecto.

Desarrollo del frontend

En una primera fase se priorizó la construcción del **frontend**, trabajando con datos simulados a través de *Mock Service Workers* (MSW). De este modo, fue posible diseñar y probar la interfaz de usuario antes de disponer de un backend funcional.

Tabla 3.1: Estimación de jornadas – Desarrollo del frontend

Tarea	Jornadas	Comentarios
Preparación del entorno de desarrollo	1	Instalación de dependencias y configuración inicial
Autenticación básica	3	Mock de login y registro en frontend
Visualización geoespacial	10	Integración de Leaflet y APIs del Catastro y SIGPAC
Pantallas "Mis parcelas" y "Detalle"	6	Diseño de UI + gestión de estados
CRUD de parcelas	2	Operaciones sobre datos simulados
Datos económicos	4	Introducción de métricas en la UI
Pantalla de inicio	3	Pantalla principal con navegación básica
Registro y login	2	Formularios y validaciones
Landing page	1	Página inicial de presentación
Total etapa frontend	**32**	-

Desarrollo del backend

En la segunda fase se implementó el **backend**, con el objetivo de sustituir los datos simulados por servicios reales y persistencia en base de datos. Esta etapa estuvo centrada en la arquitectura de microservicios con NestJS, la integración con PostgreSQL mediante TypeORM y la definición de la lógica de negocio.

Tabla 3.2: Estimación de jornadas – Desarrollo del backend

Tarea	Jornadas	Comentarios
Configuración inicial del servidor	2	Proyecto base en NestJS
Conexión a la base de datos	4	Modelado de entidades y configuración TypeORM
Endpoints de autenticación	2	Registro, login y refresh tokens
Endpoints de parcelas y recintos	4	CRUD de parcelas y recintos
Gestión de datos económicos	2	Endpoints de productividad, costes, ingresos
Total etapa backend	**14**	-

Integración final

La tercera etapa correspondió a la **integración completa** entre frontend y backend, así como la incorporación de funcionalidades avanzadas. Fue la fase que consolidó la aplicación como un prototipo funcional.

Tabla 3.3: Estimación de jornadas – Integración final

Tarea	Jornadas	Comentarios
Integración frontend-backend	4	Sustitución de mocks por API real
Procesamiento de imágenes satelitales	1	Script para recorte automático
Módulo de IA	2	Implementación del modelo
Pruebas de usabilidad y validación final	5	Iteraciones con feedback del tutor
Total etapa integración final	**12**	-

Combinando estas tres etapas junto con sus tareas y correspondientes estimacines de jornadas para completarlas, se obtiene la siguiente tabla resumen:

Tabla 3.4: Estimación de jornadas – Resumen

Etapa	Jornadas	Comentarios
Desarrollo del frontend	32	Construcción de la interfaz con datos simulados (*mocked*)
Desarrollo del backend	14	Implementación de servicios reales y persistencia
Integración y funcionalidades avanzadas	12	Conexión frontend-backend, imágenes satelitales e IA
Total del proyecto	**58**	Equivalente a 290 horas de trabajo

Cabe señalar que, aunque se establecieron estas tres fases principales, el desarrollo no fue completamente lineal. En varias ocasiones fue necesario revisar y actualizar funcionalidades previamente implementadas, por ejemplo, ajustar el CRUD de parcelas tras introducir datos económicos. Este enfoque iterativo y de retroalimentación constante garantizó que cada bloque funcional se asentara sobre una base sólida y coherente antes de avanzar al siguiente.

3.3. Estimación de recursos

La realización de este proyecto ha requerido la combinación de recursos humanos y materiales. A continuación, se detallan los principales.

Recursos humanos

Para estimar el coste que supondría externalizar el desarrollo de este proyecto, se han considerado distintos perfiles profesionales habitualmente presentes en equipos de desarrollo de software:

- **Analista de requisitos (40 horas):** encargado de recopilar y definir de manera detallada las funcionalidades de la aplicación, traduciendo las necesidades del cliente en especificaciones técnicas.
- **Desarrollador full-stack (290 horas):** responsable de la implementación del frontend, backend y de la integración de los distintos módulos en un prototipo funcional.
- **Tester (40 horas):** encargado de diseñar y ejecutar pruebas de caja negra y caja blanca, detectar errores y validar la calidad final del sistema.
- **Gestor del proyecto (30 horas):** rol de supervisión y coordinación, validando los avances y garantizando la coherencia metodológica.

Suponiendo tarifas medias de mercado en España:

- Analista de requisitos: 30 €/h.

- Desarrollador full-stack: 25 €/h.
- Tester: 20 €/h.
- Gestor del proyecto: 35 €/h.

El coste estimado sería:

Tabla 3.5: Estimación de costes de recursos humanos

Perfil	Horas	Tarifa (€/h)	Coste total
Analista de requisitos	40	30	1.200
Desarrollador full-stack	290	25	7.250
Tester / QA	40	20	800
Gestor del proyecto	30	35	1.050
Total	**400**	-	**10.300**

Cabe señalar que tanto el **modelo de inteligencia artificial** como el **script de recorte de imágenes satelitales** fueron componentes externos ya desarrollados previamente. El trabajo realizado se centró en su integración con el resto de la plataforma y en la construcción de los módulos de soporte necesarios para aprovechar sus resultados.

Recursos materiales: hardware y software

Además del esfuerzo humano, el proyecto requirió de ciertos recursos materiales. En este caso, el entorno de desarrollo se basó en un ordenador portátil MacBook Air M3 (2024), complementado con herramientas software de libre acceso.

Tabla 3.6: Estimación de costes hardware y software

Recurso	Licencia / Unidad	Coste unitario	Comentarios
MacBook Air M3	1 unidad	1.299 €	Equipo principal de desarrollo
Visual Studio Code	Licencia gratuita	0 €	Editor de código
Node.js	Licencia gratuita	0 €	Entorno de ejecución backend
PostgreSQL	Licencia gratuita	0 €	Base de datos relacional
Git + GitHub	Licencia gratuita	0 €	Control de versiones y repositorio remoto

Recurso	Licencia / Unidad	Coste unitario	Comentarios
Trello	Licencia gratuita	0 €	Tablero Kanban para gestión de tareas
APIs externas (SIGPAC, Catastro, OpenWeather)	Acceso libre	0 €	Servicios de datos oficiales y meteorológicos
Total	-	**1.299 €**	Inversión principal: hardware

Otros costes operativos

Aunque no suelen considerarse en detalle en proyectos académicos, en un escenario real también existen costes asociados al consumo energético y a la conectividad, imprescindibles para el desarrollo del trabajo.

Tabla 3.7: Estimación de otros costes operativos

Recurso	Unidad / Periodo	Coste estimado	Comentarios
Electricidad	290 h de uso del portátil	20 €	Consumo eléctrico estimado para la duración total del proyecto
Conexión a Internet	4 meses	120 €	Tarifa mensual estimada en 30 €/mes
Material de oficina (papel, bolígrafos, etc.)	Estimación global	15 €	Recursos auxiliares para anotaciones y planificación
Total	-	**195 €**	-

Así, considerando todos los recursos empleados en el desarrollo del proyecto, se calcula un presupuesto total aproximado:

Tabla 3.8: Presupuesto total aproximado

Categoría	Subtotal (€)	Comentarios
Recursos humanos	10.300	Analista, desarrollador full-stack, tester y gestor de proyecto

Categoría	Subtotal (€)	Comentarios
Recursos materiales (hardware/software)	1.299	Principalmente MacBook Air M3; el software es libre y gratuito
Otros costes operativos	195	Electricidad, internet y material de oficina
Total estimado	**11.794**	-

En la práctica profesional, resulta habitual añadir un **presupuesto de contingencia**, destinado a cubrir imprevistos como retrasos en el desarrollo, aparición de nuevos requisitos, incidencias técnicas o la necesidad de adquirir herramientas adicionales. Este margen de seguridad suele situarse entre el 10 % y el 20 % del coste total del proyecto. En este caso, se ha considerado un **15 %**, lo que incrementa el presupuesto final hasta los **13.563 €**. De esta forma, se obtiene una estimación más realista y alineada con las prácticas de gestión de proyectos en el sector del software.

Tabla 3.9: Presupuesto total aproximado con contingencia

Categoría	Subtotal (€)	Comentarios
Coste total estimado	11.794	Recursos humanos, materiales y operativos
Contingencia (15 %)	1.769	Para cubrir imprevistos y sobrecostes
Total con contingencia	**13.563**	-

El análisis global de costes sitúa la inversión estimada en torno a los **11.794–13.563 €**, donde la mayor parte corresponde a los **recursos humanos** (aproximadamente un 87 % del total). Este dato refleja que, en proyectos de software, el componente principal de gasto no suele ser el hardware ni el software —en este caso prácticamente gratuito gracias al uso de tecnologías open source—, sino la dedicación profesional de las personas implicadas.

En términos comparativos, se trata de un coste razonable para una **pyme tecnológica** o una **startup**, especialmente teniendo en cuenta que el desarrollo cubre tanto frontend como backend, integra servicios externos (SIGPAC, Catastro, OpenWeather) y sienta las bases de un módulo de inteligencia artificial aplicado a la agricultura de precisión. El bajo impacto de los recursos materiales y operativos (alrededor del 13 % del total) evidencia la eficiencia derivada del uso de **infraestructura personal y herramientas libres**.

Capítulo 4

Análisis del problema

El análisis del problema constituye una fase fundamental en el desarrollo del proyecto, ya que permite identificar con precisión las necesidades que debe cubrir el sistema y establecer las bases sobre las que se diseñará la solución tecnológica. En este capítulo se definen las características principales de la plataforma, así como los requisitos funcionales y no funcionales que guiarán su construcción.

Además, se describen los requisitos de información, es decir, los datos que la aplicación deberá gestionar para garantizar el correcto funcionamiento de sus distintas funcionalidades. Este análisis constituye el puente entre la comprensión del contexto y la planificación técnica, asegurando que la propuesta de solución esté alineada con los objetivos y expectativas iniciales del proyecto.

4.1. Características del sistema

En esta sección se presentan las propiedades y capacidades fundamentales que definen el comportamiento de la aplicación. Cada característica recoge tanto la funcionalidad principal como los atributos de calidad (rendimiento, seguridad, usabilidad...) que aseguran una experiencia coherente y alineada con los requisitos de usuario y de negocio.

Gestión de Usuarios:

- Registro
- Login
- Perfil de usuario:
  - Logout
  - Modificar
  - Eliminar

Gestión de Parcelas:

- Listado de parcelas:
  - Búsqueda
  - Ordenación
  - Filtrado
- Detalle de parcela:

    - Datos
    - Recintos
    - Imagen satelital
    - Modificar
    - Eliminar

Gestión de Recintos:

- Inspector de recintos:
    - Datos
    - Modificar datos económicos

Visualización:

- Mapa interactivo:
    - Buscar por referencia catastral
    - Agregar parcela:
        - Introducir datos (parcela)
        - Introducir datos (recinto)

Análisis:

- Viñedo:
    - Generar máscaras
    - Generar métricas
- Datos económicos:
    - Dashboard
    - Calcular métricas

Meteorología:

- Día actual (simple)
- Día actual (detalle)
- Previsión semanal

Estas propiedades se recogen en la Figura 4.1, correspondiente al **árbol de características**.

4.2. Descripción de actores

En el sistema se distinguen varios actores principales, entendidos como los usuarios o entidades externas que interactúan con la plataforma y que condicionan el diseño de sus funcionalidades:

- **Usuario registrado:** actor principal del sistema. Tiene acceso a todas las funcionalidades básicas de la aplicación: gestionar su perfil, registrar parcelas y recintos, consultar datos económicos y meteorológicos, así como visualizar información geoespacial en el mapa interactivo.

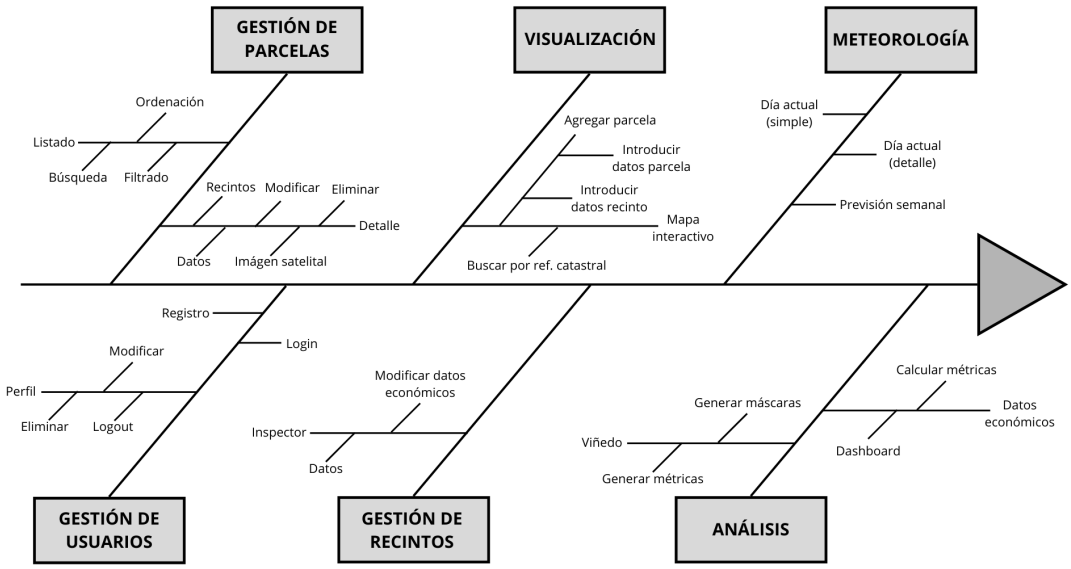

Figura 4.1: Árbol de características

- **Usuario invitado:** visitante no autenticado que accede únicamente a la *landing page* y a información básica de la plataforma. Su interacción es limitada y se centra en el registro y login para convertirse en usuario registrado.
- **Sistema SIGPAC y Catastro:** fuentes de información externas oficiales que proporcionan los datos de referencia sobre parcelas y recintos, imprescindibles para garantizar la validez de la información mostrada.
- **Servicio meteorológico externo:** proveedor de datos climáticos, utilizado para ofrecer información en tiempo real y previsiones semanales asociadas a las parcelas del usuario.
- **Módulo de inteligencia artificial:** componente encargado de procesar imágenes satelitales de viñedos para detectar hileras y calcular métricas asociadas a la productividad.
- **Script de recorte de imágenes satelitales:** servicio auxiliar que genera recortes automáticos de parcelas y recintos a partir de imágenes de satélite, devolviendo archivos optimizados para su visualización en la aplicación.

Cada actor desempeña un papel específico en el funcionamiento de la plataforma. Los usuarios constituyen el núcleo de interacción, mientras que los servicios externos y los módulos de procesamiento (IA y recorte satelital) actúan como proveedores de datos y análisis, aportando el valor añadido diferencial de la aplicación.

4.3. Requisitos de usuario

Los requisitos de usuario reflejan las necesidades y expectativas de los actores humanos que interactúan con la plataforma.

Tabla 4.1: Requisitos de usuario

Código	Descripción
RU-1	**Registro y autenticación:** los usuarios deben poder crear una cuenta fácilmente y acceder de forma segura a la aplicación mediante correo electrónico y contraseña.
RU-2	**Gestión del perfil:** los usuarios deben disponer de opciones claras para modificar o eliminar su perfil, con mensajes de confirmación y advertencia cuando las acciones sean irreversibles.
RU-3	**Acceso a datos oficiales:** los usuarios esperan que la aplicación muestre información fiable y actualizada proveniente de fuentes oficiales como SIGPAC y Catastro.
RU-4	**Visualización de parcelas y recintos:** el sistema debe ofrecer un mapa interactivo intuitivo que permita localizar, explorar y consultar fácilmente las parcelas registradas.
RU-5	**Gestión de parcelas y recintos:** los usuarios deben poder registrar nuevas parcelas y gestionar la información asociada, tanto de parcelas como de recintos.
RU-6	**Búsqueda avanzada de parcelas:** los usuarios deben poder localizar rápidamente parcelas mediante filtros por referencia catastral, ubicación geográfica u otros atributos.
RU-7	**Información meteorológica:** los usuarios deben tener acceso a datos meteorológicos actualizados y previsiones semanales asociadas a sus parcelas.
RU-8	**Gestión económica:** el sistema debe facilitar la introducción y consulta de datos económicos, así como la visualización de métricas clave (producción, ingresos, costes).
RU-9	**Integración con análisis satelital:** los usuarios esperan que las imágenes satelitales se procesen automáticamente, generando máscaras y métricas que puedan consultar sin operaciones manuales.
RU-10	**Accesibilidad y usabilidad:** la interfaz debe ser clara, responsive y usable tanto en ordenadores como en dispositivos móviles, garantizando una curva de aprendizaje mínima.
RU-11	**Privacidad y seguridad:** los usuarios deben confiar en que sus datos personales y económicos están protegidos mediante buenas prácticas de seguridad.
RU-12	**Retroalimentación y notificaciones:** el sistema debe informar al usuario de forma clara sobre el estado de sus acciones (éxito, error, procesos en curso) y ofrecer notificaciones cuando haya actualizaciones relevantes.

4.4. Casos de uso

Los casos de uso representan de forma estructurada la interacción entre los distintos actores y el sistema, describiendo las funcionalidades que éste debe ofrecer desde la perspectiva del usuario.

En esta sección se detallan los **casos de uso principales**, especificando sus actores, flujos de ejecución, precondiciones y excepciones; y se concluye con los **diagramas de casos de uso**, que proporcionan una visión gráfica de las interacciones entre los actores y el sistema, organizadas en dos vistas complementarias: gestión de usuarios y parcelas, y análisis y soporte.

Tabla 4.2: CU-01: Registro y autenticación

CU-01: Registro y autenticación	
Versión	1.0
Autor	David Pérez Hoyos
Referencias	RU-1; RF-1, RF-2, RF-3; RNF-5, RNF-6
Actor	Usuario invitado / Usuario registrado
Descripción	Permite al usuario crear una cuenta y acceder de forma segura a la aplicación.
Precondiciones	El usuario debe tener acceso a la plataforma.
Flujo normal	**Usuario:** 1. Accede a la pantalla de registro o login. 2. Introduce sus credenciales (correo y contraseña). **Sistema:** 1. Valida los datos introducidos. 2. Registra al usuario nuevo o autentica al existente. 3. Establece sesión con tokens y cookies seguras.
Postcondiciones	Usuario registrado o sesión iniciada correctamente.
Excepciones	1. Correo ya registrado → mensaje de error. 2. Credenciales inválidas → aviso al usuario.
Importancia	Alta
Observaciones	Implementar validación robusta y mensajes claros.

Tabla 4.3: CU-02: Gestión del perfil

CU-02: Gestión del perfil	
Versión	1.0
Autor	David Pérez Hoyos
Referencias	RU-2; RF-1; RNF-5, RNF-6
Actor	Usuario registrado
Descripción	Permite modificar o eliminar la información personal del perfil.
Precondiciones	El usuario debe tener sesión iniciada.
Flujo normal	**Usuario:** 1. Accede a su perfil. 2. Edita o elimina información. **Sistema:** 1. Actualiza los datos en la base de datos. 2. Confirma los cambios al usuario.
Postcondiciones	Perfil actualizado o eliminado definitivamente.
Excepciones	1. Datos inválidos → mensaje de error. 2. Eliminación cancelada → acción revertida.
Importancia	Media
Observaciones	Requiere confirmación explícita al eliminar.

Tabla 4.4: CU-03: Acceso a datos oficiales

CU-03: Acceso a datos oficiales	
Versión	1.0
Autor	David Pérez Hoyos
Referencias	RU-3; RF-4, RF-6, RF-7
Actor	Usuario registrado
Descripción	El sistema obtiene y muestra información actualizada desde fuentes oficiales (SIGPAC, Catastro).
Precondiciones	El usuario debe estar autenticado.

CU-03: Acceso a datos oficiales (cont.)	
Flujo normal	**Usuario:** 1. Solicita información de una parcela o recinto. **Sistema:** 1. Consulta los datos en los servicios oficiales. 2. Muestra la información en la interfaz de usuario.
Postcondiciones	El usuario dispone de datos oficiales actualizados.
Excepciones	1. Error en la API externa → notificación al usuario.
Importancia	Alta
Observaciones	-

Tabla 4.5: CU-04: Visualización de parcelas y recintos

CU-04: Visualización de parcelas y recintos	
Versión	1.0
Autor	David Pérez Hoyos
Referencias	RU-4; RF-6, RF-7
Actor	Usuario registrado
Descripción	Permite consultar parcelas y recintos en un mapa interactivo.
Precondiciones	El usuario debe haber registrado parcelas.
Flujo normal	**Usuario:** 1. Accede al mapa interactivo. 2. Selecciona una parcela. **Sistema:** 1. Muestra los límites de la parcela y recintos asociados. 2. Ofrece información detallada al seleccionar cada recinto.
Postcondiciones	El usuario visualiza parcelas y recintos.
Excepciones	1. Parcela no encontrada → aviso.
Importancia	Alta
Observaciones	-

Tabla 4.6: CU-05: Gestión de parcelas y recintos

CU-05: Gestión de parcelas y recintos	
Versión	1.0
Autor	David Pérez Hoyos
Referencias	RU-5; RF-4, RF-7
Actor	Usuario registrado
Descripción	Permite gestionar datos de parcelas y recintos.
Precondiciones	El usuario debe tener sesión iniciada.
Flujo normal	**Usuario:** 1. Accede a la sección "Mis parcelas". 2. Edita una parcela/recinto. **Sistema:** 1. Guarda la información en la base de datos. 2. Confirma la operación.
Postcondiciones	Datos de parcelas y recintos actualizados.
Excepciones	1. Datos inválidos → mensaje de error.
Importancia	Alta
Observaciones	Mantener integridad con datos oficiales.

Tabla 4.7: CU-06: Búsqueda avanzada de parcelas

CU-06: Búsqueda avanzada de parcelas	
Versión	1.0
Autor	David Pérez Hoyos
Referencias	RU-6; RF-5, RF-6
Actor	Usuario registrado
Descripción	Permite localizar parcelas mediante filtros y criterios avanzados.
Precondiciones	El usuario debe tener parcelas registradas.

CU-06: Búsqueda avanzada de parcelas (cont.)	
Flujo normal	**Usuario:** 1. Introduce filtros (ej. referencia catastral). **Sistema:** 1. Ejecuta la búsqueda. 2. Muestra resultados en lista.
Postcondiciones	El usuario encuentra parcelas según sus filtros.
Excepciones	1. Ningún resultado → mensaje de aviso.
Importancia	Media
Observaciones	Implementar búsqueda eficiente.

Tabla 4.8: CU-07: Información meteorológica

CU-07: Información meteorológica	
Versión	1.0
Autor	David Pérez Hoyos
Referencias	RU-7; RF-9
Actor	Usuario registrado
Descripción	Muestra el estado y previsiones meteorológicas asociadas a las parcelas.
Precondiciones	El usuario debe tener parcelas registradas.
Flujo normal	**Usuario:** 1. Consulta la meteorología de una parcela. **Sistema:** 1. Solicita datos al servicio meteorológico externo. 2. Muestra información actual y previsiones.
Postcondiciones	El usuario dispone de datos meteorológicos actualizados.
Excepciones	1. Error en el servicio externo → notificación.
Importancia	Media
Observaciones	-

Tabla 4.9: CU-08: Gestión económica

CU-08: Gestión económica	
Versión	1.0
Autor	David Pérez Hoyos
Referencias	RU-8; RF-10
Actor	Usuario registrado
Descripción	Permite introducir y consultar datos económicos por recinto.
Precondiciones	El usuario debe tener parcelas y recintos registrados.
Flujo normal	**Usuario:** 1. Introduce datos económicos (producción, costes, ingresos). **Sistema:** 1. Almacena y calcula métricas básicas.
Postcondiciones	Datos económicos disponibles para análisis.
Excepciones	1. Datos incompletos o erróneos → aviso.
Importancia	Alta
Observaciones	Permite análisis agregado posterior.

Tabla 4.10: CU-09: Integración con análisis satelital

CU-09: Integración con análisis satelital	
Versión	1.0
Autor	David Pérez Hoyos
Referencias	RU-9; RF-8
Actor	Usuario registrado
Descripción	El sistema procesa automáticamente imágenes satelitales para generar máscaras y métricas.
Precondiciones	El usuario debe tener parcelas registradas.
Flujo normal	**Sistema:** 1. Obtiene imágenes satelitales de la parcela. 2. Aplica el script de recorte. 3. Ejecuta el modelo IA (viñedos). 4. Devuelve métricas procesadas al usuario.

CU-09: Integración con análisis satelital (cont.)	
Postcondiciones	Usuario visualiza imágenes recortadas y métricas.
Excepciones	1. Imagen no disponible → notificación.
Importancia	Alta
Observaciones	Componente crítico de valor añadido.

Tabla 4.11: CU-10: Accesibilidad y usabilidad

CU-10: Accesibilidad y usabilidad	
Versión	1.0
Autor	David Pérez Hoyos
Referencias	RU-10; RNF-9
Actor	Usuario registrado
Descripción	Garantiza que la interfaz sea usable en diferentes dispositivos.
Precondiciones	Usuario autenticado o invitado en landing page.
Flujo normal	**Usuario:** 1. Accede a la aplicación desde un dispositivo cualquiera. **Sistema:** 1. Ajusta la interfaz de forma responsive. 2. Aplica pautas de accesibilidad.
Postcondiciones	Usuario puede utilizar la aplicación sin barreras.
Excepciones	-
Importancia	Media
Observaciones	Compatible con navegadores modernos.

Tabla 4.12: CU-11: Privacidad y seguridad

CU-11: Privacidad y seguridad	
Versión	1.0
Autor	David Pérez Hoyos
Referencias	RU-11; RNF-5, RNF-6, RNF-7
Actor	Usuario registrado
Descripción	Protege los datos personales y económicos del usuario.

CU-11: Privacidad y seguridad (cont.)	
Precondiciones	Usuario con datos registrados en la aplicación.
Flujo normal	**Sistema:** 1. Encripta credenciales y datos sensibles. 2. Aplica validaciones y sanitización en inputs. 3. Protege endpoints con autenticación segura.
Postcondiciones	Datos almacenados y accesibles solo para el usuario.
Excepciones	-
Importancia	Muy alta
Observaciones	Clave para la confianza de los usuarios.

Tabla 4.13: CU-12: Retroalimentación y notificaciones

CU-12: Retroalimentación y notificaciones	
Versión	1.0
Autor	David Pérez Hoyos
Referencias	RU-12; RF-11; RNF-10
Actor	Usuario registrado
Descripción	El sistema informa sobre el estado de operaciones y muestra notificaciones relevantes.
Precondiciones	Usuario autenticado.
Flujo normal	**Usuario:** 1. Ejecuta una acción (ej. guardar datos). **Sistema:** 1. Muestra mensaje de confirmación, error o proceso en curso. 2. Genera notificaciones cuando existan cambios relevantes.
Postcondiciones	Usuario informado del estado de sus acciones.
Excepciones	-
Importancia	Media
Observaciones	Notificaciones deben ser claras y no intrusivas.

Las Figuras 4.2 y 4.3 corresponden al **diagrama de casos de uso** del sistema, dividido en dos vistas complementarias.

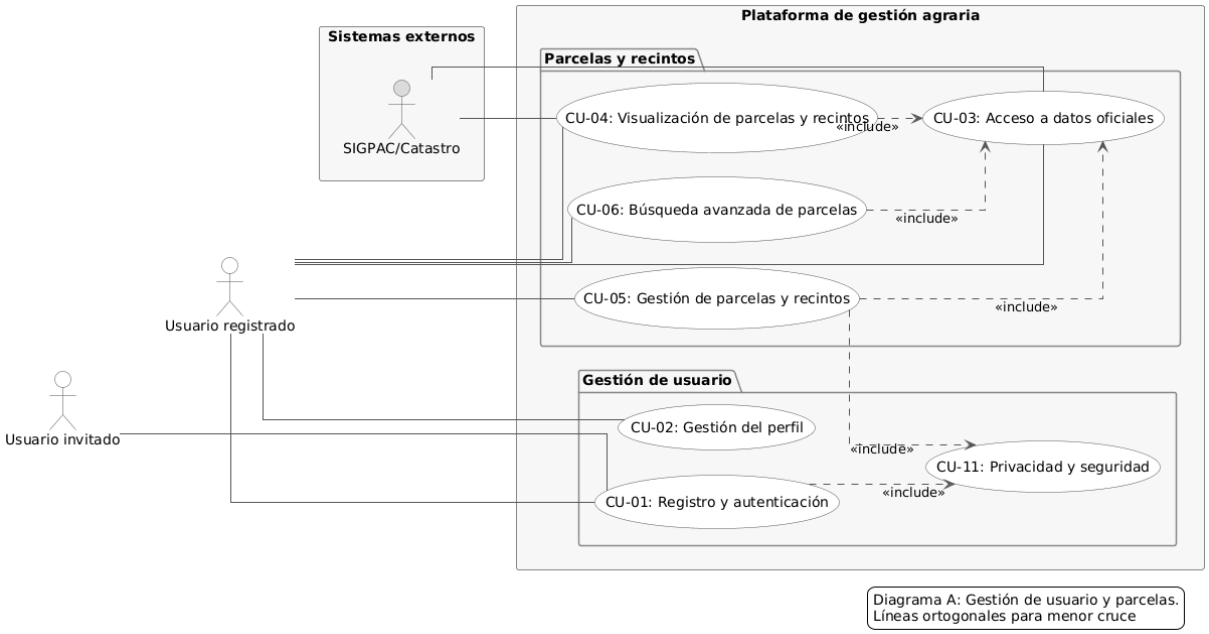

Figura 4.2: Diagrama de casos de uso — Gestión de usuario y parcelas

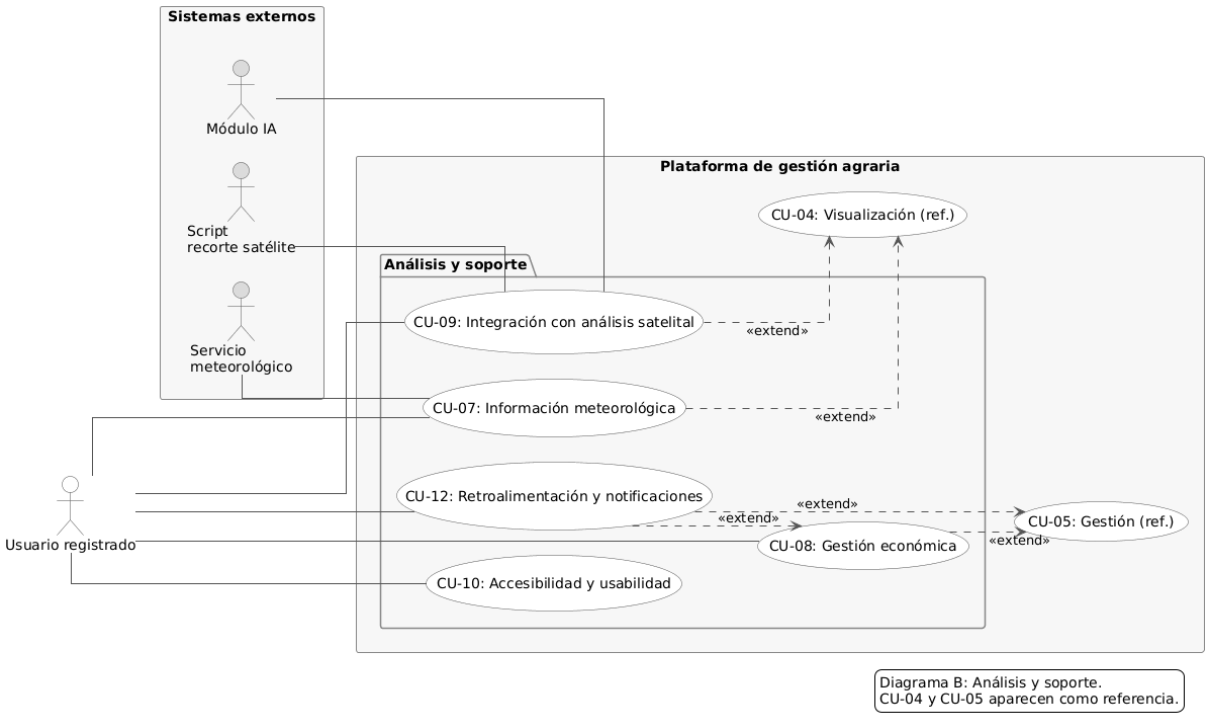

Figura 4.3: Diagrama de casos de uso — Análisis y soporte

4.5. Requisitos funcionales

A continuación se describen los requisitos funcionales de la plataforma, es decir, las capacidades y comportamientos específicos que el sistema debe proporcionar para dar soporte a las operaciones de usuarios y procesos clave y garantizar el cumplimiento de los objetivos del proyecto.

Tabla 4.14: Requisitos funcionales del sistema

Código	Descripción
RF-1	**Gestión de usuarios:** el sistema debe permitir a los usuarios registrarse, consultar, modificar sus datos personales y eliminar su cuenta de forma irreversible.
RF-2	**Inicio de sesión:** el sistema debe ofrecer un formulario de acceso donde los usuarios introduzcan correo electrónico y contraseña, validando credenciales y mostrando errores claros en caso de datos inválidos.
RF-3	**Autenticación y control de sesiones:** la aplicación contará con un sistema de autenticación mediante tokens JWT, utilizando la estrategia de *refresh tokens*.
RF-4	**Gestión de parcelas:** cada usuario podrá asignar a su perfil múltiples parcelas. Cada parcela tendrá asociada su referencia catastral, identificación SIGPAC y otros datos relevantes.
RF-5	**Filtros y ordenación:** la lista de parcelas del usuario debe poder filtrarse y ordenarse según diferentes criterios.
RF-6	**Visor de parcelas:** la aplicación contará con un mapa interactivo en el que se mostrarán todas las parcelas junto con su información actualizada, obtenida de fuentes oficiales como el SIGPAC o el Catastro.
RF-7	**Subdivisión de parcelas en recintos:** siguiendo las especificaciones oficiales, las parcelas se dividen en recintos. El sistema debe manifestar este hecho y exponer la información actualizada específica de cada recinto.
RF-8	**Análisis satelital y predicciones automáticas:** para cada parcela y recinto se obtendrá su imagen satelital y, en caso de viñedos, se ejecutará un análisis mediante un modelo entrenado de inteligencia artificial.
RF-9	**Información meteorológica:** el sistema deberá mostrar información meteorológica actualizada de diferentes localidades, según la ubicación del usuario y parcela.
RF-10	**Información económica:** el usuario podrá introducir datos económicos personalizados (productividad (kg), precio de venta (€/kg), o precio de compra (€/kg)) para cada recinto de cada parcela asignada.

Código	Descripción
RF-11	**Análisis económico:** la aplicación deberá mostrar diferentes análisis económicos utilizando métricas relevantes que faciliten la toma de decisiones del usuario.

En la Tabla 4.15 se muestra la correspondencia establecida entre cada RU y los RF que lo materializan en la aplicación.

Tabla 4.15: Matriz de trazabilidad entre RU-RF

RU	Descripción resumida	RF relacionados
RU-1	Registro y autenticación de usuarios	RF-1, RF-2, RF-3
RU-2	Gestión del perfil de usuario	RF-1
RU-3	Acceso a datos oficiales (SIGPAC, Catastro)	RF-4, RF-6, RF-7
RU-4	Visualización de parcelas y recintos	RF-6, RF-7
RU-5	Gestión de parcelas y recintos (alta, edición)	RF-4, RF-7
RU-6	Búsqueda avanzada de parcelas	RF-5, RF-6
RU-7	Información meteorológica actualizada	RF-9
RU-8	Gestión económica (producción, costes, ingresos)	RF-10, RF-11
RU-9	Integración con análisis satelital (imágenes y métricas)	RF-8
RU-10	Accesibilidad y usabilidad (interfaz clara, responsive)	RF-6, RF-9, RF-10, RF-11
RU-11	Privacidad y seguridad de datos	RF-2, RF-3, RF-10
RU-12	Retroalimentación y notificaciones al usuario	RF-2, RF-6, RF-10, RF-11

4.6. Requisitos no funcionales

Los requisitos no funcionales establecen las calidades y restricciones que debe cumplir el sistema más allá de sus funciones concretas. En este proyecto, garantizar estos atributos es clave para ofrecer una plataforma ágil, confiable y fácil de evolucionar, capaz de atender tanto a técnicos como a gestores agrarios bajo condiciones reales de uso.

Tabla 4.16: Requisitos no funcionales del sistema

Código	Descripción
RNF-1	**Rendimiento:** las operaciones deben procesarse en menos de 300 ms bajo condiciones normales de uso.

Código	Descripción
RNF-2	**Disponibilidad:** el sistema deberá alcanzar un mínimo del 99.5 % de tiempo operativo mensual.
RNF-3	**Escalabilidad:** deberá soportar aumentos de carga mediante escalado horizontal (nuevas instancias) sin cambios en el código ni interrupciones perceptibles.
RNF-4	**Mantenibilidad:** el código seguirá estándares (linting, formateo, estructura modular) que faciliten su lectura, pruebas y evolución futura.
RNF-5	**Seguridad en datos sensibles:** las contraseñas y otros posibles datos sensibles deberán protegerse mediante encriptación y nunca almacenarse en formato plano.
RNF-6	**Seguridad en *endpoints*:** todas las rutas estarán protegidas contra accesos no autorizados, con manejo de errores y excepciones controladas.
RNF-7	**Seguridad en *inputs*:** los datos de entrada serán validados y saneados para prevenir inyecciones, XSS u otros ataques.
RNF-8	**Integridad:** se garantizará la consistencia transaccional en la base de datos (ACID) y mediante restricciones (FK, unicidad).
RNF-9	**Usabilidad:** la interfaz será intuitiva y *responsive*, adaptándose a distintos dispositivos y tamaños de pantalla.
RNF-10	**Feedback:** se proporcionará retroalimentación clara en operaciones de carga, éxito y error; los mensajes seguirán un estilo uniforme.
RNF-11	**Portabilidad:** la aplicación funcionará correctamente en navegadores modernos (Chrome, Firefox, Safari, Edge) sin requerir *plugins* adicionales.
RNF-12	**Separación de entornos:** se distinguirán y configurarán los entornos de desarrollo y producción, definiendo variables de entorno y configuraciones específicas para cada uno.
RNF-13	**Robustez:** el sistema recuperará o degradará funcionalidades de forma controlada ante fallos parciales, sin colapsar el flujo de trabajo.
RNF-14	**Extensibilidad:** la arquitectura modular permitirá incorporar nuevas características o integraciones externas sin alterar el núcleo existente.
RNF-15	**Concurrencia:** el sistema deberá soportar al menos 200 usuarios activos simultáneamente sin degradación significativa del rendimiento.
RNF-16	**Tiempo de despliegue:** el despliegue de una nueva versión en producción no deberá exceder de 15 minutos, incluyendo migraciones de esquema y reinicio de servicios.

4.7. Requisitos de información

El sistema requiere gestionar y almacenar distintos conjuntos de datos, provenientes tanto de los propios usuarios como de fuentes oficiales externas. Estos datos constituyen la base para las funcionalidades de visualización, análisis económico y soporte a la toma de decisiones.

Tabla 4.17: Requisitos de información del sistema

Código	Descripción
RI-1	**Datos de usuario:** el sistema debe almacenar de forma segura las credenciales de acceso (correo electrónico y contraseña encriptada), así como información personal básica y datos de perfil editables.
RI-2	**Datos de parcelas:** la aplicación debe gestionar la información identificativa de cada parcela (referencia catastral, provincia, municipio, polígono, número de parcela), junto con atributos personalizados definidos por el usuario (nombre, notas, descripción).
RI-3	**Datos de recintos:** el sistema debe registrar las subdivisiones oficiales de cada parcela, asociando atributos como código SIGPAC, superficie, uso del suelo y variedad de cultivo.
RI-4	**Datos económicos:** la aplicación debe permitir la introducción, almacenamiento y consulta de datos económicos por recinto (variedad cultivada, producción en kg, costes, ingresos y precios de venta/compra).
RI-5	**Datos meteorológicos:** el sistema debe integrar (pero no almacenar) información meteorológica en tiempo real y previsiones semanales obtenidas de servicios externos, vinculándola a la ubicación de las parcelas.
RI-6	**Datos satelitales:** la aplicación debe gestionar imágenes descargadas automáticamente desde servicios externos, utilizándolas para recortar parcelas y generar máscaras a nivel de recinto.
RI-7	**Resultados de análisis IA:** el sistema debe almacenar y mostrar las métricas generadas por el modelo de inteligencia artificial aplicado a viñedos, así como las máscaras generadas.

Capítulo 5

Diseño de la solución

El diseño de la solución constituye la fase en la que se traduce el análisis previo en una propuesta técnica concreta. A partir de los requisitos identificados —funcionales, no funcionales, de usuario e información— se definen las estructuras, modelos y mecanismos que permitirán implementar la aplicación de forma coherente, eficiente y escalable. Este capítulo describe la arquitectura general del sistema, los principales diagramas de interacción, el modelo de datos y las decisiones de diseño adoptadas en el frontend, el backend y los módulos transversales como la autenticación y la gestión de servicios externos.

El objetivo es establecer una guía clara y justificada de cómo se materializa la plataforma, garantizando trazabilidad con los objetivos iniciales y ofreciendo un marco sólido para la fase de implementación.

5.1. Arquitecturas

La arquitectura del sistema se concibe bajo un enfoque **cliente-servidor** con separación clara entre frontend y backend. Esta decisión permite aislar la lógica de negocio del cliente, mejorar la escalabilidad y favorecer la reutilización de servicios.

Arquitectura lógica

La arquitectura lógica organiza el sistema en tres capas bien diferenciadas que separan responsabilidades y facilitan la evolución del software:

- **Capa de presentación (frontend):** implementada en React con Vite y Tailwind. Gestiona la interacción con el usuario, el renderizado de mapas con Leaflet y la composición de dashboards. La gestión del *estado global* se centraliza mediante **Redux**, lo que permite mantener sincronizados los datos de sesión, las parcelas, los recintos y la información económica en toda la interfaz. Esta capa se comunica con el backend a través de peticiones HTTP y, además, realiza *llamadas directas* a servicios públicos externos como SIGPAC y Catastro para mostrar geometrías oficiales.

- **Capa de negocio (backend):** desarrollada en NestJS con una estructura modular (controladores y servicios). Orquesta la lógica de usuarios, parcelas, recintos y datos económicos; aplica seguridad (JWT con cookies HttpOnly) y coordina los servicios auxiliares de recorte de imágenes y análisis IA.

- **Capa de datos:** persistencia relacional en PostgreSQL (usuarios, parcelas, recintos, datos económicos) y almacenamiento de resultados de análisis (máscaras/medidas) para su consulta posterior.

La Figura 5.1 representa el **diagrama de arquitectura lógica** del sistema.

Este diseño por capas permite cumplir los requisitos no funcionales de mantenibilidad, escalabilidad y seguridad: el frontend permanece desacoplado del dominio, el backend concentra reglas y políticas de acceso, y la base de datos garantiza integridad referencial. El uso de Redux en el frontend aporta consistencia y facilita la comunicación entre componentes, reduciendo la complejidad de gestión del estado.

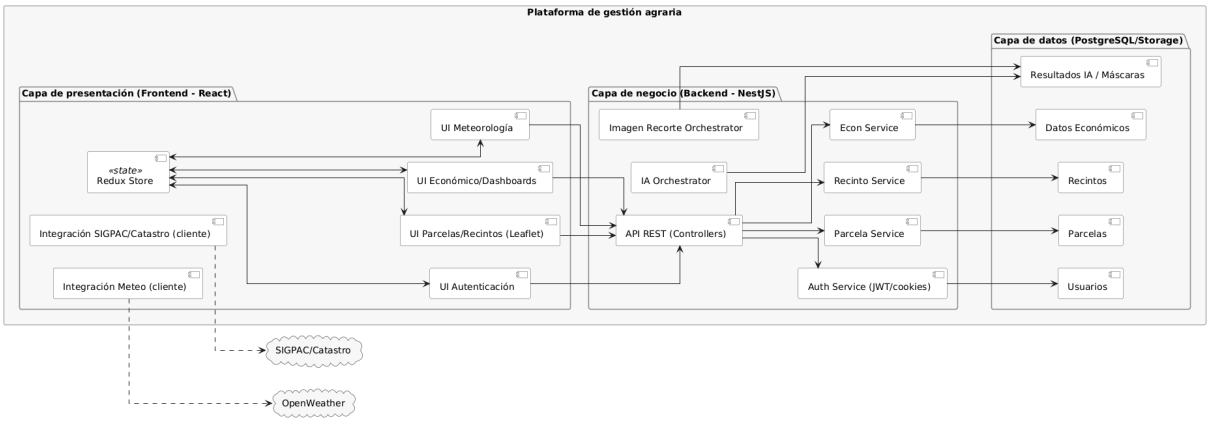

Figura 5.1: Diagrama de arquitectura lógica del sistema

Arquitectura física

La arquitectura física representa cómo se despliega el sistema en una infraestructura tecnológica concreta. Aunque el desarrollo se llevó a cabo en un entorno local (ordenador personal), se plantea una configuración de despliegue en producción orientada a la nube:

- **Cliente:** los usuarios acceden a la aplicación a través de un navegador web, tanto desde ordenadores como desde dispositivos móviles. El frontend, compilado con *Vite*, puede alojarse en un servidor web o distribuirse mediante una *Content Delivery Network* (CDN), lo que asegura baja latencia y disponibilidad global.

- **Servidor de aplicación:** el backend desarrollado en *NestJS* se despliega en un servidor dedicado o contenedores Docker gestionados en plataformas cloud (ej. AWS, Azure, Google Cloud). Este módulo concentra la lógica de negocio, gestiona peticiones REST y coordina el acceso a la base de datos y a servicios externos.

- **Base de datos:** PostgreSQL se ejecuta en un servidor independiente, preferiblemente como servicio gestionado (*Database-as-a-Service*), lo que facilita la replicación, la seguridad, la escalabilidad horizontal y la automatización de copias de seguridad.

- **Servicios externos:** los sistemas oficiales (SIGPAC, Catastro) y el proveedor meteorológico (OpenWeather) se consumen a través de peticiones HTTP seguras. Adicionalmente, los módulos de **recorte de imágenes satelitales** y de **análisis IA**

de viñedos operan como servicios auxiliares, invocados desde el backend y responsables de aportar los datos procesados.

La Figura 5.2 representa el **diagrama de arquitectura física** del sistema.

Esta separación física permite distribuir la carga, escalar los componentes críticos de forma independiente y reforzar la seguridad mediante el aislamiento entre cliente, servidor de aplicación y base de datos.

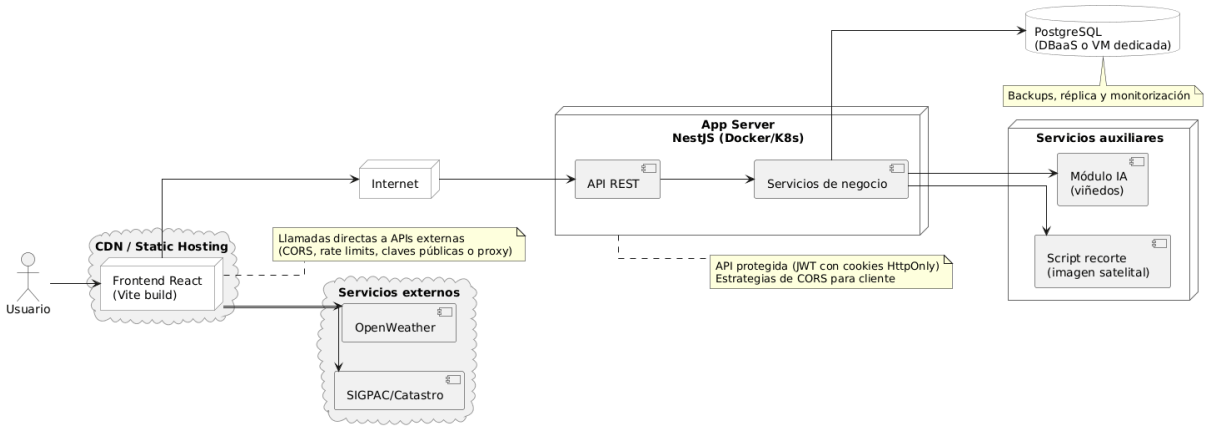

Figura 5.2: Diagrama de arquitectura física del sistema

5.2. Diagramas de secuencia

En esta sección se ilustran distintos **diagramas de secuencia** que muestran cómo interactúan los actores, el frontend (React + Redux Toolkit), el backend (NestJS) y los servicios externos durante la ejecución de los principales casos de uso del sistema. Estos diagramas reflejan el flujo temporal de mensajes y operaciones, aportando una visión clara de la dinámica de la aplicación.

DS-1: Autenticación de usuarios

Este diagrama muestra los procesos de **login**, **registro** y **logout**. Se aprecia cómo el frontend utiliza RTK Query (`authApi`) y los `slices` de Redux para gestionar tokens de sesión, actualizando la *store* en función de la respuesta del backend.

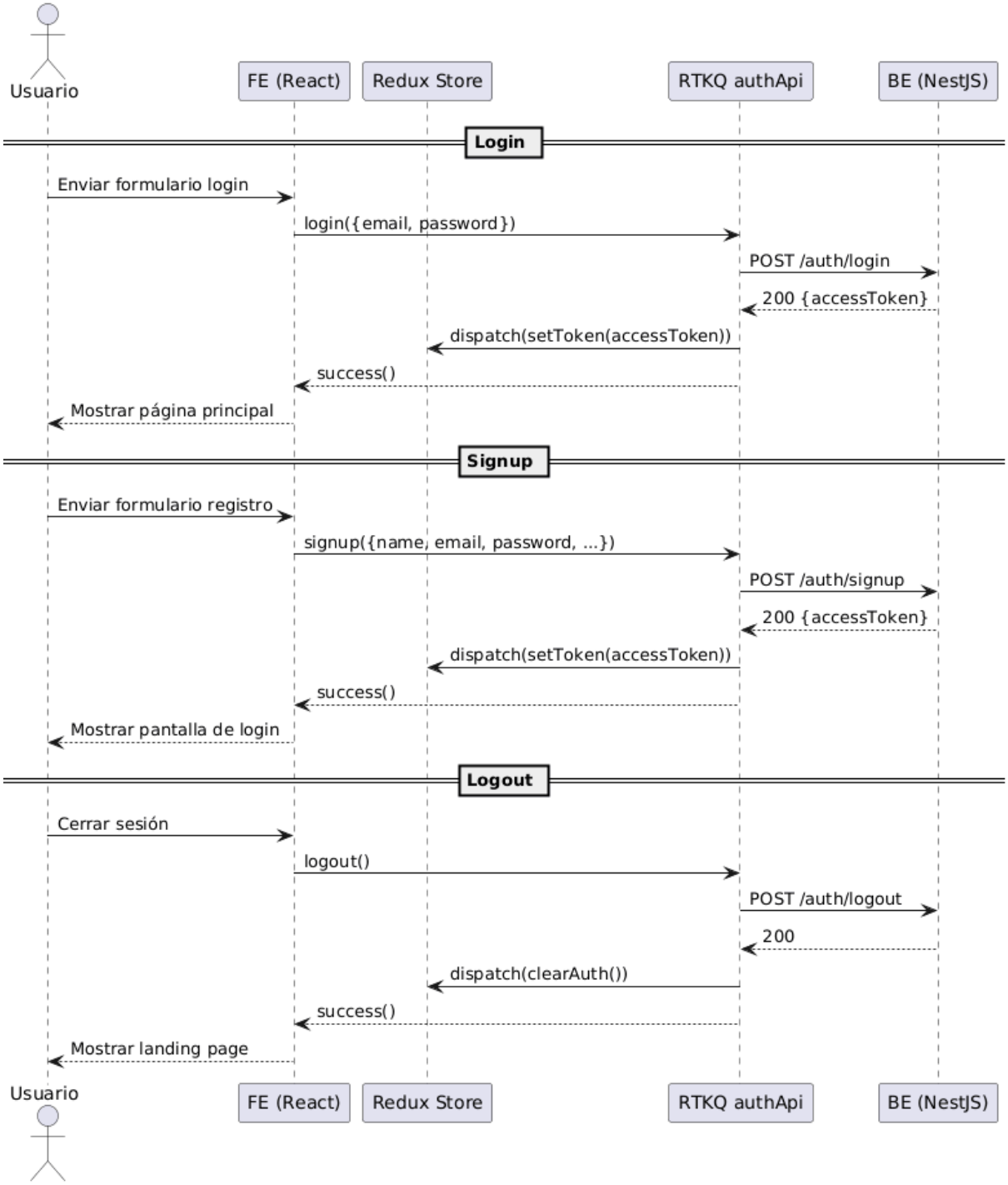

Figura 5.3: Diagrama de secuencia — Login, signup y logout

DS-2: Re-autenticación automática

Este diagrama refleja la lógica implementada en `baseQueryWithReauth`. Cuando una petición retorna `401 Unauthorized`, el sistema intenta refrescar el token con el endpoint correspondiente y reintenta la petición original.

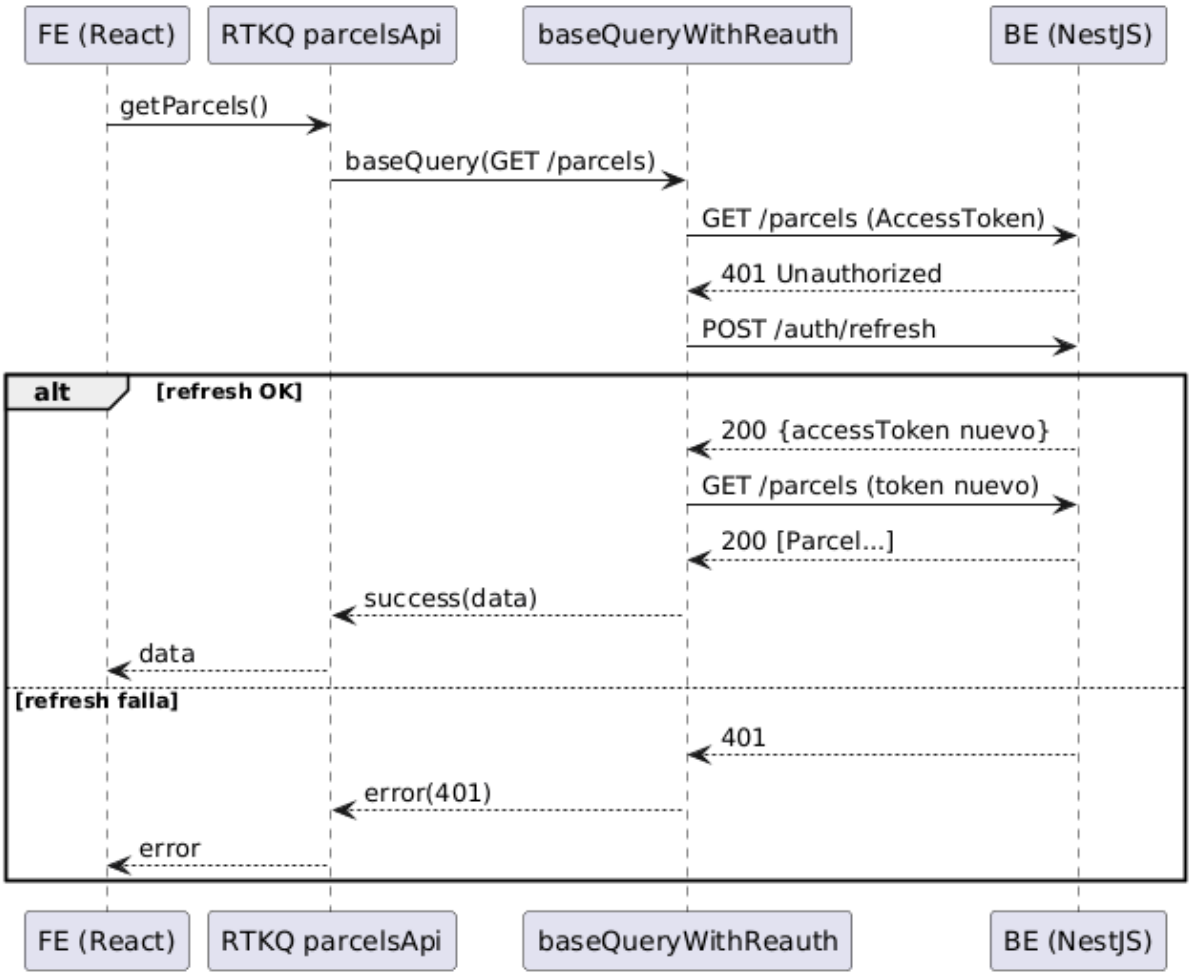

Figura 5.4: Diagrama de secuencia — Re-autenticación automática

DS-3: Listado y gestión de parcelas

Se ilustra la consulta del listado de parcelas mediante `getParcelsSummary` y cómo las operaciones de alta, edición o borrado invalidan la caché con `invalidatesTags`, provocando un refresco automático de datos.

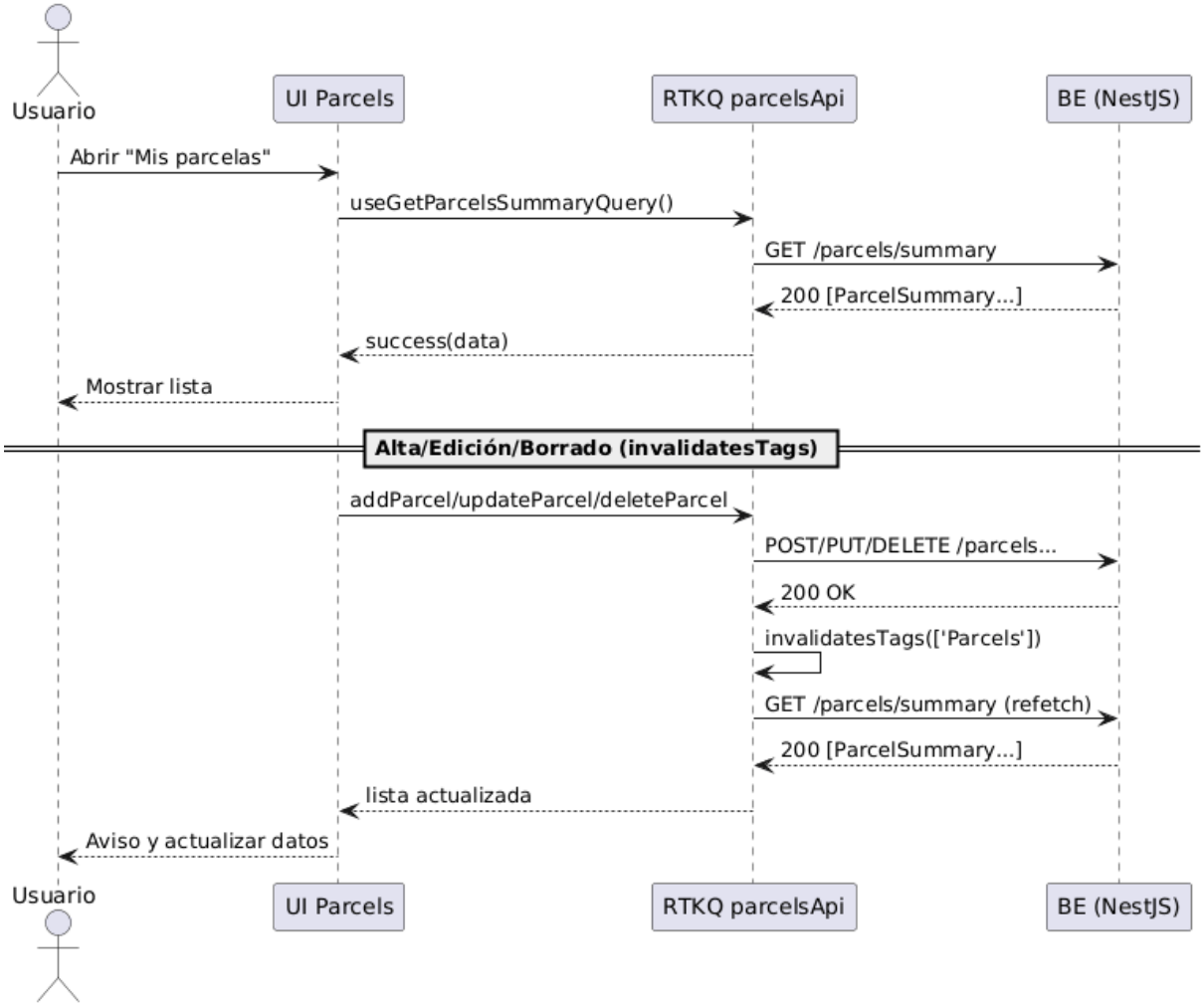

Figura 5.5: Diagrama de secuencia — Listado y gestión de parcelas

DS-4: Actualización de datos económicos

Este diagrama describe la edición de información económica de un recinto (`updateRecinto`), la propagación de cambios en la base de datos y la invalidación de caché que actualiza automáticamente la vista del usuario.

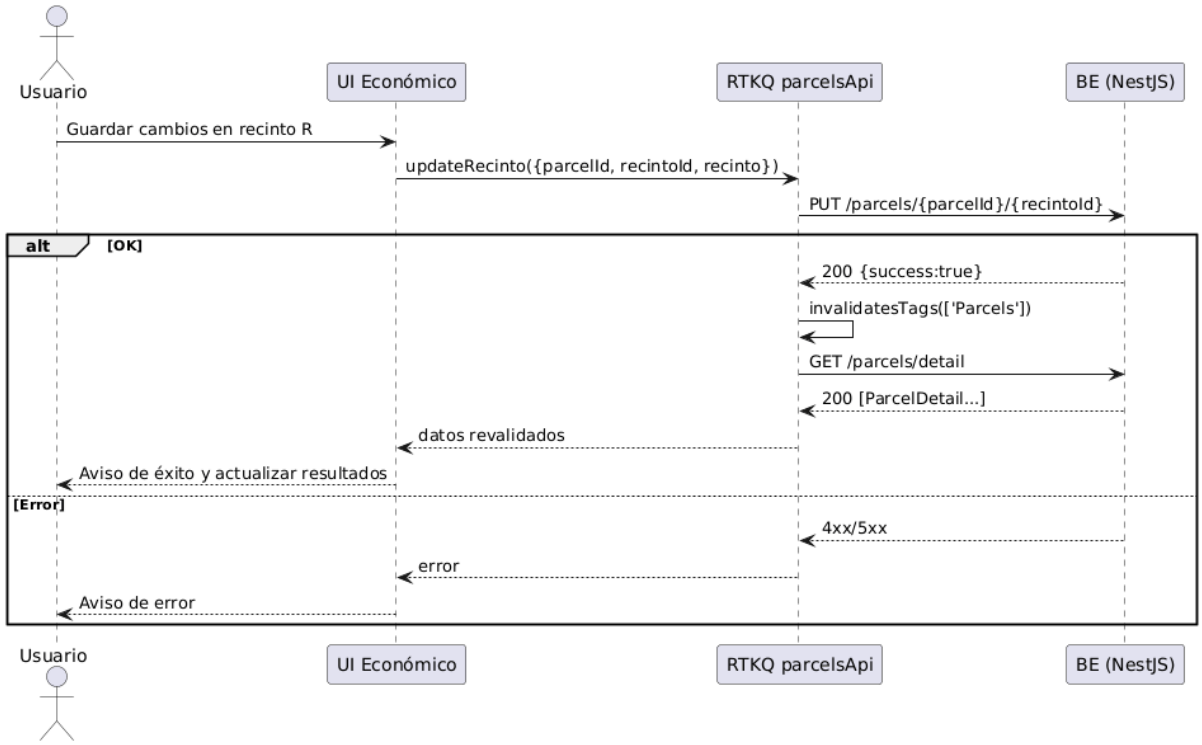

Figura 5.6: Diagrama de secuencia — Actualización de datos económicos

DS-5: Generación de imágenes y análisis IA

Se muestran los flujos de dos procesos avanzados: la **generación de imágenes recortadas** de parcelas y el **análisis de viñedos con IA**. Ambos se gestionan a través del backend, que orquesta servicios auxiliares.

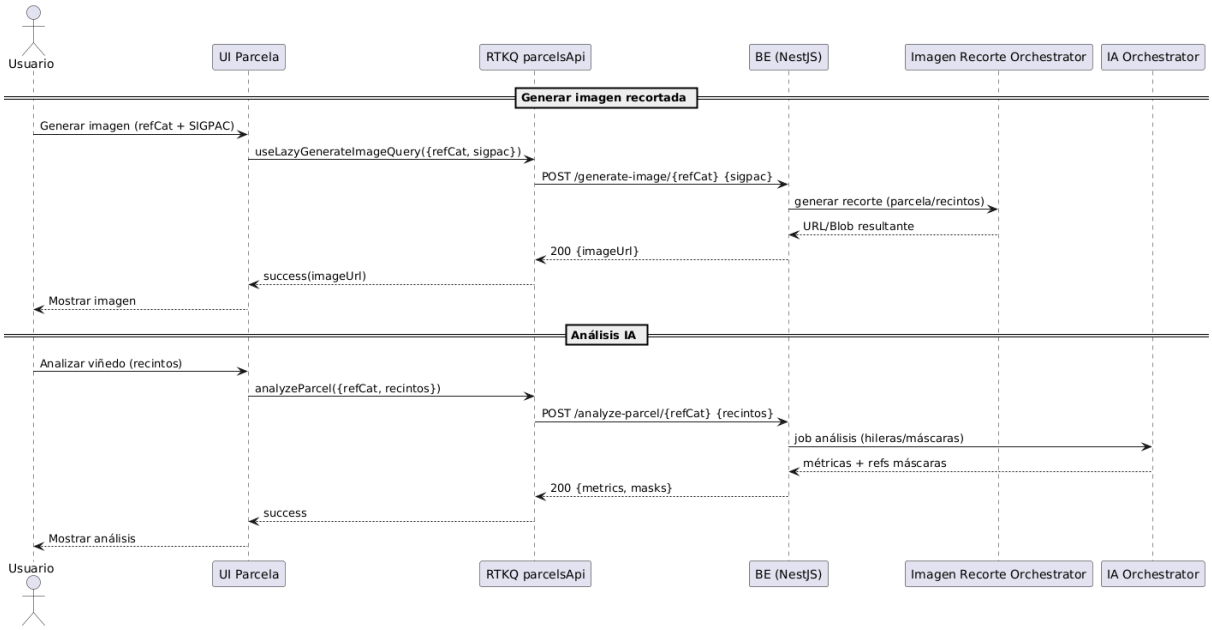

Figura 5.7: Diagrama de secuencia — Generación de imágenes y análisis con IA

DS-6: Integración con servicios externos

Este diagrama representa las llamadas directas desde el frontend a **SIGPAC/Catastro** y **Open-Meteo**, utilizando los servicios `visorApi` y `weatherApi`. Los datos obtenidos se muestran en Leaflet o en módulos de meteorología.

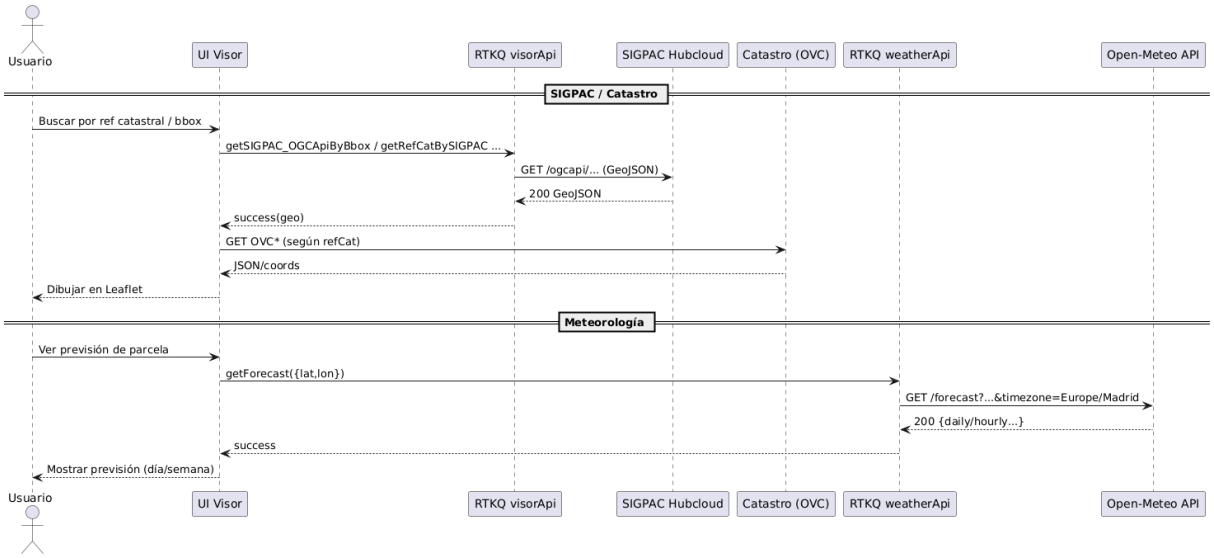

Figura 5.8: Diagrama de secuencia — Integración con SIGPAC/Catastro y meteorología

DS-7: Gestión del perfil de usuario

Este diagrama muestra las operaciones principales sobre el perfil: **actualizar datos** y **eliminar cuenta**. En el primer caso, los cambios se envían al backend mediante `updateUser` y, tras confirmarse en la base de datos, se actualiza la *store* de Redux. En el segundo, la mutación `delete` elimina al usuario en el servidor y borra su sesión en la aplicación.

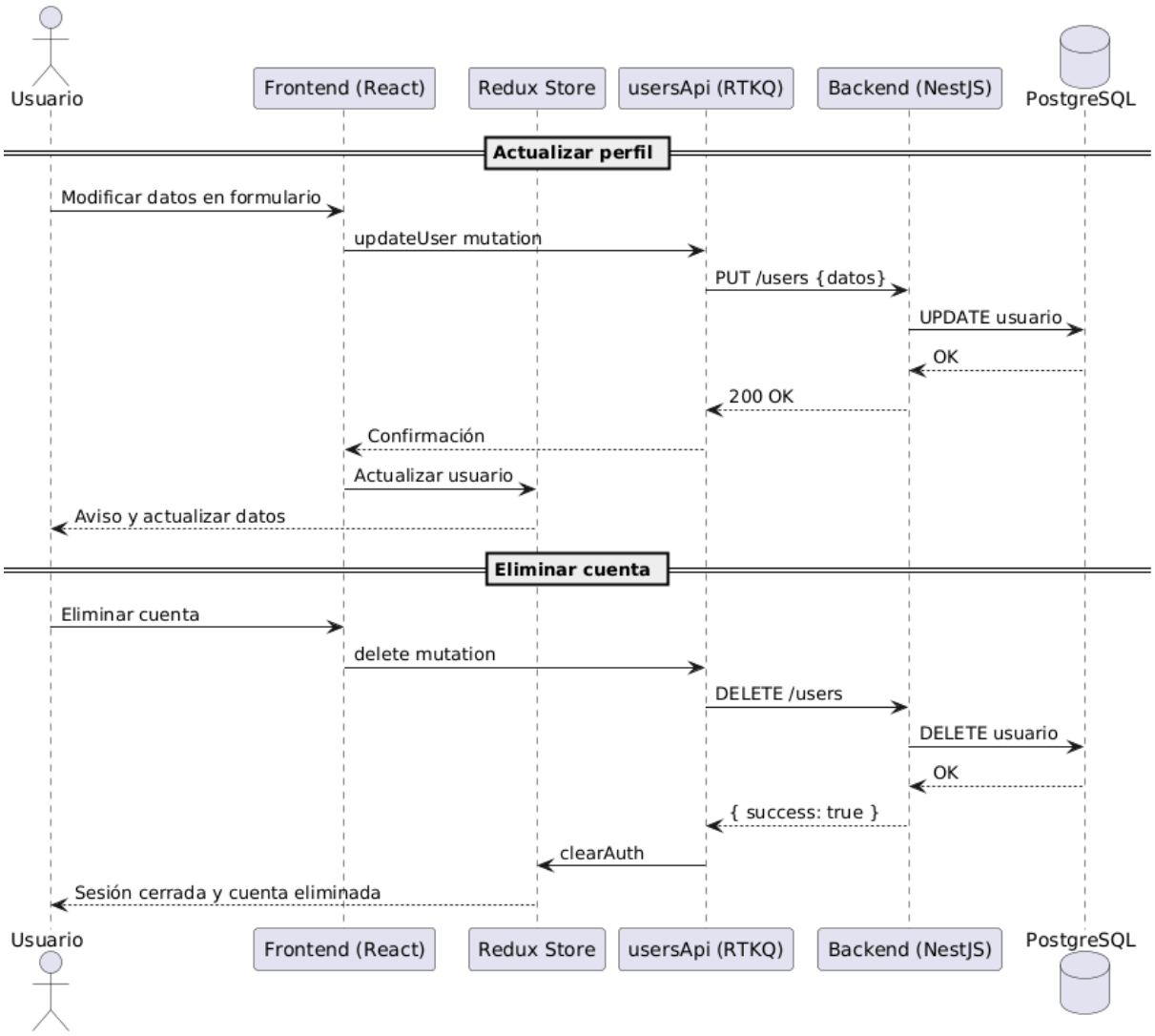

Figura 5.9: Diagrama de secuencia — Gestión del perfil de usuario

5.3. Modelado de datos

El modelado de datos es una parte fundamental en el desarrollo de la aplicación, ya que permite estructurar y organizar la información de forma coherente y optimizada para su posterior uso. En este proyecto se ha utilizado **PostgreSQL** como sistema gestor de base de datos, y su diseño se ha representado mediante un **diagrama entidad–relación lógico** que refleja las tablas, atributos, claves primarias y foráneas, así como las relaciones entre ellas.

En la Figura 5.10 se muestra dicho esquema, generado con la herramienta *DBeaver*, que sirve como guía visual para comprender la estructura y las dependencias existentes entre las distintas entidades.

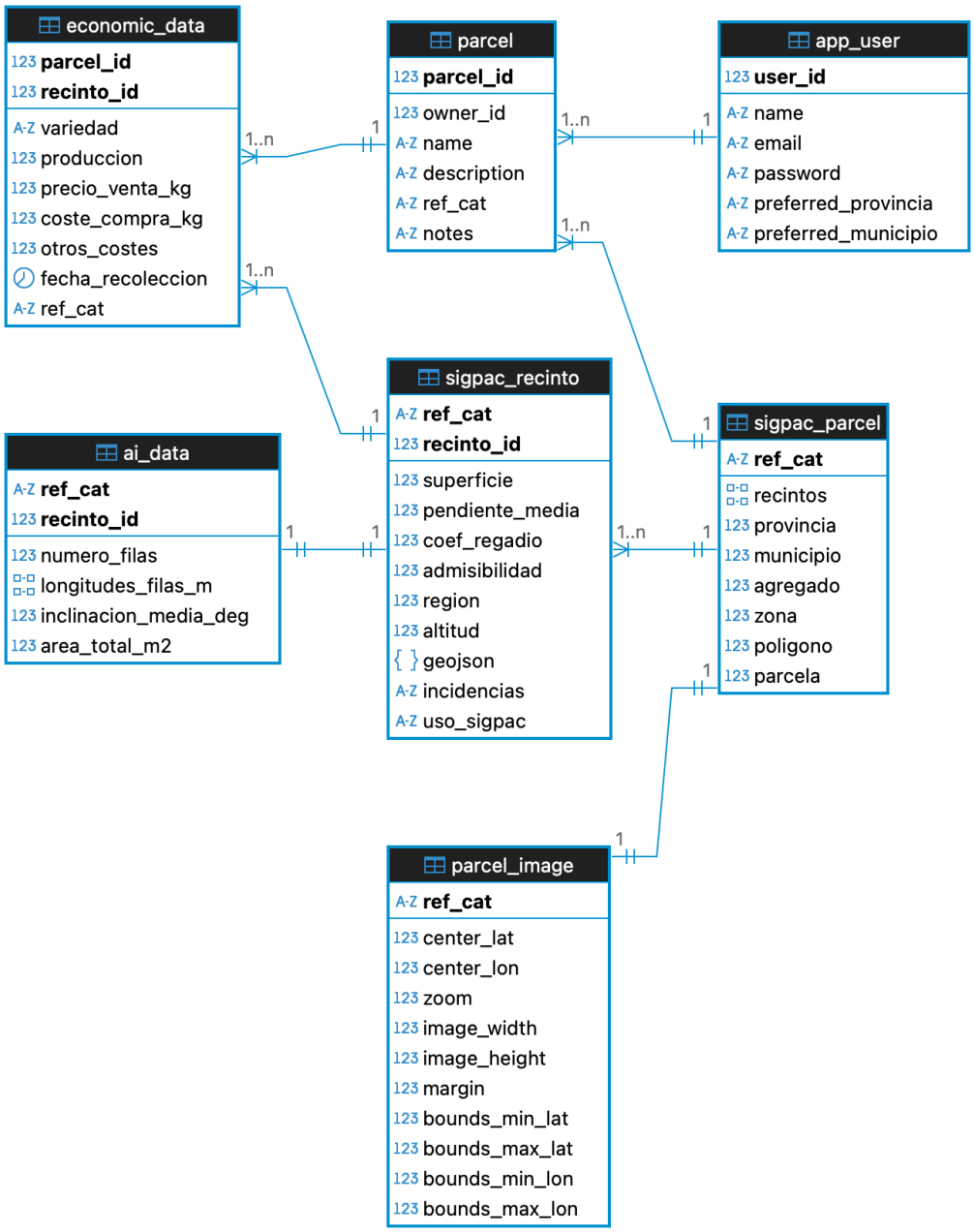

Figura 5.10: Diagrama entidad–relación lógico de la base de datos

Diccionario de datos

A continuación se detallan las entidades principales de la base de datos PostgreSQL. Para cada tabla se especifican sus campos, tipo de dato, nulabilidad, claves y una breve descripción.

Tabla `app_user`

Tabla 5.1: Diccionario de datos — `app_user`

Campo	Tipo	Nulos	Clave	Descripción
user_id	integer	NO	PK	Identificador único del usuario.
name	varchar(50)	NO		Nombre completo del usuario.
email	varchar(255)	NO	UQ	Correo electrónico, único en el sistema.
password	text	NO		Contraseña encriptada.
preferred_provincia	varchar(255)	NO		Provincia seleccionada por defecto.
preferred_municipio	varchar(255)	NO		Municipio seleccionado por defecto.

Tabla `parcel`

Tabla 5.2: Diccionario de datos — `parcel`

Campo	Tipo	Nulos	Clave	Descripción
parcel_id	integer	NO	PK	Identificador único de la parcela.
owner_id	integer	NO	FK	Referencia al usuario propietario (`app_user.user_id`).
name	varchar(50)	NO		Nombre asignado a la parcela.
description	varchar(300)	NO		Breve descripción de la parcela.
ref_cat	varchar	NO	FK	Referencia catastral asociada a `sigpac_parcel`.
notes	varchar	SÍ		Notas o comentarios adicionales.

Integridad:

- `owner_id` → `app_user.user_id` (CASCADE).
- `ref_cat` → `sigpac_parcel.ref_cat` (CASCADE).

Tabla `sigpac_parcel`

Tabla 5.3: Diccionario de datos — `sigpac_parcel`

Campo	Tipo	Nulos	Clave	Descripción
ref_cat	varchar(20)	NO	PK	Referencia catastral de la parcela.
recintos	integer[]	NO		Lista de identificadores de recintos que pertenecen a la parcela.
provincia	integer	NO		Código de la provincia.

municipio	integer	NO	Código del municipio.
agregado	integer	NO	Código de agregado SIGPAC.
zona	integer	NO	Código de zona SIGPAC.
poligono	integer	NO	Número de polígono.
parcela	integer	NO	Número de parcela.

Tabla `sigpac_recinto`

Tabla 5.4: Diccionario de datos — `sigpac_recinto`

Campo	Tipo	Nulos	Clave	Descripción
ref_cat	varchar(20)	NO	PK*, FK	Referencia catastral (`sigpac_parcel.ref_cat`).
recinto_id	integer	NO	PK*	Identificador del recinto dentro de la parcela.
superficie	numeric(12,2)	SÍ		Superficie en hectáreas.
pendiente_media	numeric(12,2)	SÍ		Pendiente media del recinto.
coef_regadio	numeric(12,2)	SÍ		Coeficiente de regadío.
admisibilidad	numeric(12,2)	SÍ		Nivel de admisibilidad.
region	integer	SÍ		Región SIGPAC.
altitud	numeric(12,2)	SÍ		Altitud sobre el nivel del mar.
geojson	json	NO		Geometría del recinto en formato GeoJSON.
incidencias	varchar	SÍ		Incidencias reportadas.
uso_sigpac	varchar	SÍ		Uso SIGPAC asignado al recinto.

Integridad:

- `ref_cat` → `sigpac_parcel.ref_cat` (CASCADE).

Tabla `parcel_image`

Tabla 5.5: Diccionario de datos — `parcel_image`

Campo	Tipo	Nulos	Clave	Descripción
ref_cat	varchar	NO	PK, FK	Referencia catastral (`sigpac_parcel.ref_cat`).
center_lat	double precision	NO		Latitud del centro de la imagen.

center_lon	double precision	NO	Longitud del centro de la imagen.
zoom	integer	NO	Nivel de zoom usado para la captura.
image_width	integer	NO	Anchura en píxeles.
image_height	integer	NO	Altura en píxeles.
margin	double precision	NO	Margen aplicado alrededor de la parcela.
bounds_min_lat	double precision	NO	Límite inferior de latitud.
bounds_max_lat	double precision	NO	Límite superior de latitud.
bounds_min_lon	double precision	NO	Límite inferior de longitud.
bounds_max_lon	double precision	NO	Límite superior de longitud.

Integridad:

- `ref_cat` → `sigpac_parcel.ref_cat` (CASCADE).

Tabla `economic_data`

Tabla 5.6: Diccionario de datos — `economic_data`

Campo	Tipo	Nulos	Clave	Descripción
parcel_id	integer	NO	PK*, FK	Parte de la PK*; referencia a `parcel.parcel_id`.
recinto_id	integer	NO	PK*	Parte de la PK* compuesta (`parcel_id`, `recinto_id`).
ref_cat	varchar(20)	NO	FK	Clave foránea a `sigpac_recinto`.
variedad	varchar(50)	NO		Variedad cultivada.
produccion	double precision	NO		Producción en kilogramos.
precio_venta_kg	numeric(12,2)	NO		Precio de venta en €/kg.
coste_compra_kg	numeric(12,2)	NO		Coste de compra en €/kg.
otros_costes	numeric(12,2)	NO		Otros costes asociados.
fecha_recoleccion	date	NO		Fecha de recolección.

Integridad:

- `parcel_id` → `parcel.parcel_id` (CASCADE).
- `(ref_cat, recinto_id)` → `sigpac_recinto` (CASCADE).

Tabla `ai_data`

Tabla 5.7: Diccionario de datos — `ai_data`

Campo	Tipo	Nulos	Clave	Descripción
ref_cat	varchar(20)	NO	PK, FK	Referencia catastral del recinto.
recinto_id	integer	NO	PK, FK	Identificador del recinto.
numero_filas	integer	NO		Número de filas detectadas.
longitudes_filas_m	numeric(12,2)[]	NO		Longitud de cada fila (en metros).
inclinacion_media_deg	numeric(12,2)	NO		Inclinación media en grados.
area_total_m2	numeric(12,2)	NO		Área total del recinto en metros cuadrados.

Integridad:

- `(ref_cat, recinto_id)` → `sigpac_recinto` (CASCADE).

Convenciones:

- **PK**: Clave primaria; **FK**: clave foránea. **PK*** indica clave primaria compuesta.
- Las FKs con *ON DELETE CASCADE* garantizan limpieza de datos derivados al eliminar entidades padres.

5.4. Diseño de la interfaz de usuario

La interfaz de usuario se ha diseñado siguiendo un enfoque empresarial, basado en el uso de **paneles o tarjetas** que agrupan la información en bloques diferenciados. Este patrón de diseño facilita la lectura, permite al usuario localizar rápidamente la información de interés y proporciona coherencia visual en todas las pantallas de la aplicación.

Cada panel representa un tipo de contenido específico, como previsiones meteorológicas, información de parcelas, datos económicos o elementos de navegación.

Landing page

La *landing page* actúa como puerta de entrada a la aplicación. En ella se muestran paneles informativos de carácter general y accesos directos que invitan al usuario a registrarse o iniciar sesión. Su diseño sencillo busca transmitir una primera impresión clara y accesible.

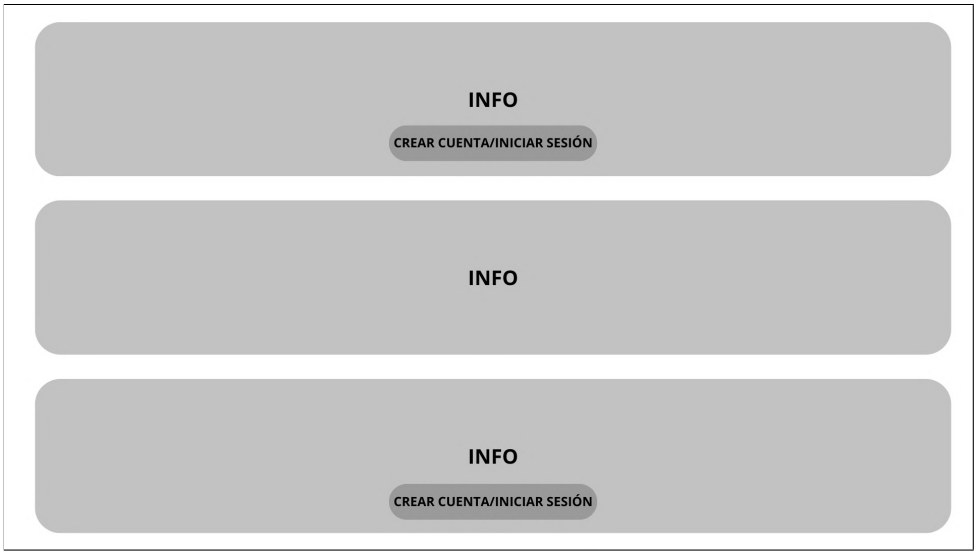

Figura 5.11: Diseño de la interfaz — Landing page

Home

La pantalla de inicio organiza la información en distintos paneles, que actúan como un *dashboard* personalizado. Aquí el usuario encuentra un resumen de su actividad, accesos a funcionalidades principales y vistas rápidas de datos relevantes. Se trata del punto de partida para la navegación dentro del sistema.

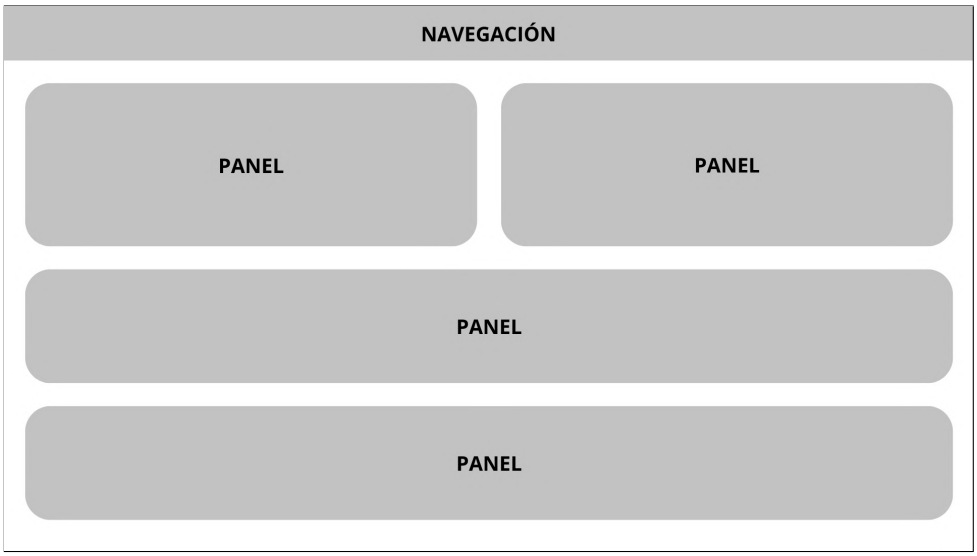

Figura 5.12: Diseño de la interfaz — Home

Mis Parcelas

Esta vista muestra en formato de tarjetas todas las parcelas registradas por el usuario, con opciones de filtro y búsqueda. Cada tarjeta ofrece información básica, y si el usuario hace click, se le redirige al detalle de la parcela, cuyo diseño sigue la misma estructura que la mostrada en la Figura 5.12.

Figura 5.13: Diseño de la interfaz — Mis Parcelas

Visor de Parcelas

El visor, implementado sobre la librería *Leaflet*, permite al usuario explorar parcelas y recintos en un mapa interactivo. Este ocupa la mayor parte de la pantalla; mostrando en los laterales dos paneles con diferentes opciones y detalles.

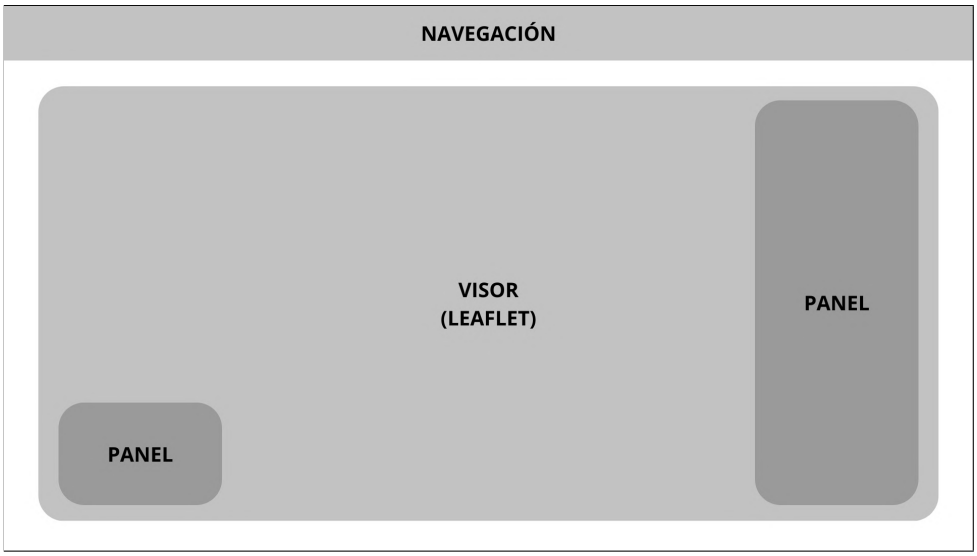

Figura 5.14: Diseño de la interfaz — Visor de Parcelas

Perfil de Usuario

En esta pantalla el usuario puede consultar y modificar sus datos personales. Se trata de un solo panel donde se recogen todos los datos y opciones necesarias.

Figura 5.15: Diseño de la interfaz — Perfil de Usuario

Paleta de colores utilizada

El diseño visual de la aplicación se ha basado en una paleta sencilla y consistente, que favorece la usabilidad y mantiene una apariencia profesional. Se han definido tres grupos de colores:

- **Colores principales**: Amarillo (`yellow-300`) y gris oscuro (`gray-700`), que visualmente parece azul marino. Estos se utilizan en elementos clave de la interfaz, como la barra de navegación, botones principales y títulos destacados. El amarillo aporta viveza y contraste, mientras que el gris oscuro otorga seriedad y legibilidad.

- **Colores secundarios**: Blanco y gris claro (`gray-300`). Se aplican como fondos neutros para no sobrecargar la interfaz y mantener un equilibrio visual. Favorecen la separación entre secciones y ayudan a que los colores principales resalten más.

- **Colores auxiliares**: Se emplean en tarjetas y paneles específicos, como las vistas meteorológicas (azul) o los resúmenes económicos (verde), aportando contexto visual inmediato al usuario.

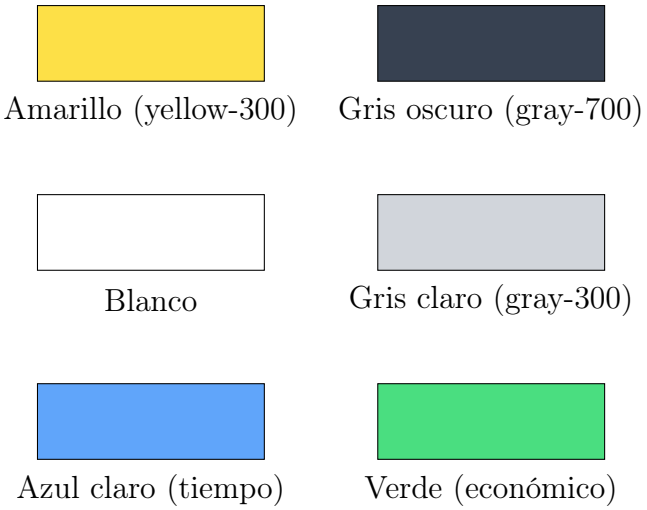

Figura 5.16: Paleta de colores de la aplicación

Capítulo 6

Desarrollo e implementación

En este capítulo se describe el proceso de construcción de la solución propuesta, desde la implementación de la interfaz de usuario en el *frontend* hasta la lógica de negocio y la persistencia de datos en el *backend*.

El objetivo es mostrar cómo las decisiones de diseño planteadas en el capítulo anterior se materializan en código, tecnologías y librerías concretas, garantizando así una aplicación funcional, escalable y mantenible.

6.1. Frontend

El frontend se ha desarrollado en **React** con **TypeScript** (JavaScript con tipado), gestionando el estado global mediante **Redux Toolkit** y consultas de datos con **RTK Query**. Para el enrutamiento se ha empleado **React Router**, y para la simulación de peticiones HTTP durante el desarrollo se ha utilizado **MSW (Mock Service Worker)**.

El diseño de la interfaz se ha implementado siguiendo los principios de **usabilidad** y **consistencia visual**, apoyándose en bibliotecas como **Tailwind CSS** para el estilo y **Leaflet** para la visualización geoespacial de las parcelas.

Cada componente ha sido desarrollado como unidad modular y reutilizable, facilitando la mantenibilidad y la evolución futura del sistema.

6.1.1. Estructura de carpetas del frontend

La carpeta `src/` contiene toda la lógica del frontend. Dentro de ella se encuentran las siguientes subcarpetas:

- `api/`: lógica común de consultas HTTP, incluyendo el `baseQueryWithReauth`.
- `commons/`: componentes reutilizables como modales, alertas, cabeceras o spinners.
- `components/`: componentes específicos de la aplicación, como tarjetas de parcela, paneles meteorológicos o paneles económicos.
- `hooks/`: hooks personalizados que abstraen lógica de negocio (ej. `useAuth`, `useParcels`).
- `mocks/`: simulación del backend con MSW, incluyendo `handlers.ts` y `server.ts`.
- `pages/`: vistas principales de la aplicación (Home, Parcels, Profile, Visor).

- `redux/`: `slices` y `store` de Redux Toolkit.
- `router/`: configuración de rutas, incluyendo rutas privadas y públicas.
- `services/`: definición de endpoints de RTK Query para cada dominio (auth, parcels, users, weather, etc.).
- `types/`: definición de interfaces y tipos TypeScript para garantizar tipado estricto en datos.
- `utils/`: funciones auxiliares (ej. validadores de formularios).

Además, el frontend cuenta con dos archivos de configuración de entorno en la raíz del proyecto, que permiten definir el backend al que se conectará la aplicación según el entorno. De esta forma, el cambio entre entornos es automático y no requiere modificar el código fuente, únicamente el archivo de configuración correspondiente.

- `.env.development`:

  ```
  VITE_API_URL=http://localhost:3000
  ```

  Utilizado en el entorno local de desarrollo. La API apunta al backend ejecutándose en la misma máquina.

- `.env.production`:

  ```
  VITE_API_URL=https://mi-backend-en-produccion.com
  ```

  Utilizado al desplegar la aplicación en producción. Permite que el frontend apunte directamente al servidor remoto.

Por último, en `src/config.ts` se centralizan estas variables para que el resto de la aplicación acceda a ellas de forma uniforme:

```
export const BACKEND_URL = import.meta.env.VITE_API_URL || 'http://localhost:3000';

export const isDev = import.meta.env.DEV;
export const isProd = import.meta.env.PROD;

export const API_URL = `${BACKEND_URL}/api`;
export const IMAGES_URL = `${BACKEND_URL}/images`;
export const PARCELS_IMAGES_URL = `${IMAGES_URL}/parcels`;
```

Este archivo ofrece una capa de abstracción que evita la dispersión de rutas y constantes por el código, y facilita mantener un mismo punto de configuración para todas las rutas del backend.

A continuación se describen los contenidos de algunas de las carpetas más importantes del proyecto:

Componentes comunes (`commons/`)

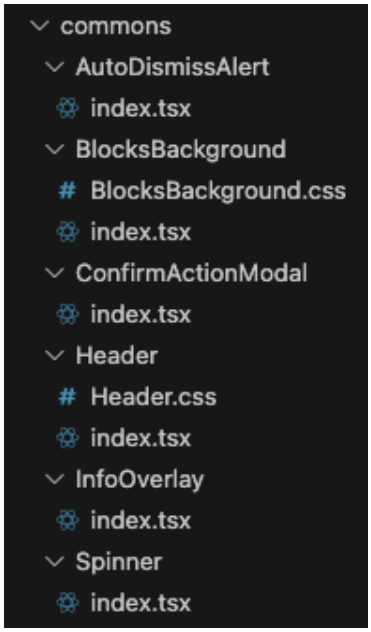

Figura 6.1: Implementación Frontend — Componentes comunes

En esta carpeta se agrupan los componentes reutilizables y transversales a la aplicación. Estos no dependen de un dominio concreto (parcelas, meteorología, usuario, etc.), sino que pueden ser empleados en diferentes vistas y contextos. A continuación, se describen los principales:

- **AutoDismissAlert**: componente de alerta temporal que muestra mensajes de notificación (éxito, error, advertencia) y se cierra automáticamente tras unos segundos. Resulta fundamental para dar feedback inmediato al usuario tras operaciones como guardar cambios o iniciar sesión.
- **BlocksBackground**: fondo decorativo utilizado en ciertas pantallas, compuesto por bloques de color. Aporta coherencia visual y mantiene la estética corporativa de la aplicación.
- **ConfirmActionModal**: ventana modal que solicita confirmación explícita antes de realizar operaciones críticas, como eliminar una parcela o borrar la cuenta de usuario. Mejora la usabilidad y la seguridad al prevenir acciones accidentales.
- **Header**: cabecera común presente en todas las páginas de la aplicación. Incluye la barra de navegación, enlaces a las secciones principales y, en su caso, accesos directos al perfil del usuario.
- **InfoOverlay**: capa de superposición con información adicional o contextual. Se emplea, por ejemplo, para mostrar instrucciones, ayudas o descripciones sobre un elemento sin interrumpir el flujo principal.
- **Spinner**: indicador de carga animado que aparece durante la ejecución de peticiones asíncronas o procesos que requieren tiempo de espera. Ayuda a transmitir al usuario que el sistema está procesando la operación.

Estos componentes facilitan la consistencia visual y funcional en toda la aplicación, reduciendo la duplicidad de código y mejorando la mantenibilidad.

Gestión del estado global (`redux/`)

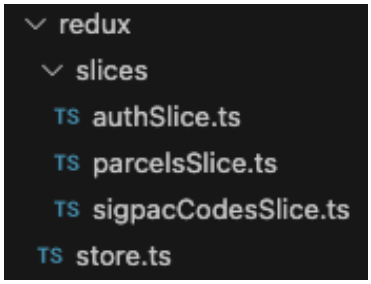

Figura 6.2: Implementación Frontend — Gestión del estado global

La gestión del estado global de la aplicación se implementa con **Redux Toolkit (RTK)**, lo que permite centralizar la información compartida entre múltiples componentes y mantener la aplicación sincronizada.

- La `store` es el estado global completo.
- Cada `slice` es una parte de ese estado (ej. usuario, parcelas, códigos SIGPAC).
- Los `reducers` son las funciones que modifican esas partes del estado.

Servicios (`services/`)

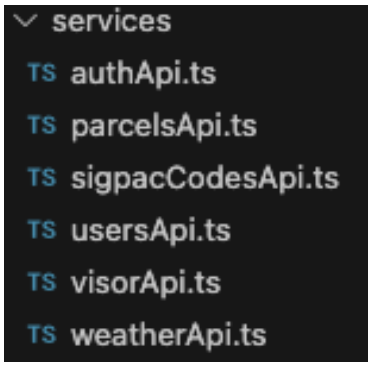

Figura 6.3: Implementación Frontend — Servicios

La carpeta `services/` agrupa los servicios de **RTK Query**, que definen los distintos **endpoints** del backend y de las APIs externas (ver Anexo A). Cada archivo contiene la configuración de un dominio concreto, facilitando la organización y el consumo de datos en el frontend:

- `authApi.ts`: operaciones relacionadas con autenticación y gestión de sesión.
- `parcelsApi.ts`: consultas y modificaciones sobre las parcelas del usuario.

- `sigpacCodesApi.ts`: acceso a los códigos de referencia del SIGPAC (provincias, municipios, usos, incidencias).
- `usersApi.ts`: gestión de datos de usuario.
- `visorApi.ts`: obtención de recintos, parcelas y referencias catastrales desde el servicio SIGPAC.
- `weatherApi.ts`: obtención de previsiones meteorológicas diarias y horarias.

6.1.2. Autenticación con *refresh tokens* en el frontend

La aplicación implementa un esquema de autenticación **JWT** + ***refresh tokens*** apoyado en Redux Toolkit y RTK Query. El objetivo es mantener sesiones de usuario seguras y persistentes con mínima fricción para el usuario.

Se diferencian dos tipos de *tokens*:

- **Tokens de acceso** (*access tokens*): El *access token* es un JWT de corta duración (p. ej., 15 minutos) que contiene la identidad del usuario y sus permisos. Es el que se adjunta en cada petición al backend mediante el encabezado `Authorization: Bearer <token>`. Su corta vida útil minimiza riesgos en caso de robo, ya que caduca rápidamente.
- **Tokens de refresco** (*refresh tokens*): El *refresh token* tiene una duración mucho mayor (días o semanas) y su única función es solicitar nuevos tokens de acceso cuando estos expiran. Por seguridad, se almacena en una cookie con la bandera `HttpOnly`, inaccesible desde el *frontend* para evitar ataques de tipo XSS.

En la implementación se distinguen las siguientes **piezas principales**:

- `baseQueryWithReauth` (`src/api/baseQueryWithReauth.ts`): añade el *access token* como cabecera a todas las peticiones hechas por el frontend. Ante un código de error `401 (Unauthorized)`, intenta renovar el token llamando a `/auth/refresh` incluyendo, si existe, la cookie `HttpOnly` que contiene el *refresh token*. Si la renovación tiene éxito, **reintenta** la petición original; si falla, limpia el estado de sesión.
- `authSlice` (`src/redux/slices/authSlice.ts`): estado global de autenticación (en Redux) que guarda si el usuario está identificado, junto con sus datos básicos. Además, tiene métodos como `setToken` (establecer token), `setUser` (establecer usuario) y `clearAuth` (borrar usuario y token, es decir, limpiar la sesión).
- `useAuthInit` (`src/hooks/useAuthInit.ts`): proceso de inicio que, al cargar la aplicación, comprueba si el usuario tiene una sesión activa y la restaura en caso afirmativo.

Flujo operativo

1. **Arranque de la aplicación**: al iniciar, el estado de sesión en Redux está vacío, por lo que la app llama al endpoint `/auth/refresh` para comprobar si existe una sesión activa.

   a) Si el servidor devuelve un `accessToken` válido, se decodifica para obtener la información básica del usuario (`id`, `name`, `email`, `preferredProvincia`, `preferredMunicipio`).
   b) Estos datos, junto con el token, se guardan en el estado global, dejando al usuario en estado autenticado.

   Si no se devuelve un token válido, o hay error en la petición, se limpia el estado (`clearAuth`) y la aplicación queda como no autenticada.

2. **Navegación dentro de la aplicación**: cada vez que el usuario interactúa, gracias a `baseQueryWithReauth` las peticiones al backend incluyen el *access token* actual.
   a) Si el token está caducado, el sistema automáticamente hace una llamada a `/auth/refresh`, enviando la cookie de sesión (*refresh token*).
   b) Si el servidor responde con un nuevo token válido, este se guarda en el estado global y la petición original se repite sin que el usuario lo note.
   c) Si la renovación falla, se limpia el estado y el usuario pasa a estar no autenticado.

3. **Gestión de carga y errores**: durante este proceso, la aplicación controla estados como `isLoading`, `isError` o `isAuthenticated`. Con ellos se muestra un overlay de carga o error (`InfoOverlay`) y se adapta la interfaz, por ejemplo ocultando la barra de navegación cuando no hay sesión activa.

En la Figura 6.4 se ilustra este proceso en forma de diagrama de secuencia.

6.1.3. Inicialización de los datos

Al iniciar la aplicación, no solo se valida si el usuario dispone de una sesión activa, sino que también se cargan los datos básicos necesarios para que la experiencia sea fluida desde el primer momento. Este proceso se lleva a cabo de forma paralela mediante los *hooks* `useAuthInit` (Figura 6.5) y `useLoadAppData` (Figuras 6.6 y 6.7), que combinan la comprobación de sesión con la carga de información inicial.

Si la autenticación mediante `useAuthInit` (que sigue el proceso descrito en 6.1.2) es exitosa, el *slice* `authSlice` se nutre con los datos del usuario y el *access token*. A continuación, se rellenan otros *slices* del estado global, fundamentales para el funcionamiento del sistema: `parcelsSlice` y `sigpacCodesSlice`.

- `parcelsSlice`: mantiene en memoria datos básicos de las parcelas del usuario. Esto permite que pantallas como *Mis Parcelas* se carguen de forma inmediata, sin depender de nuevas llamadas al *backend*.
- `sigpacCodesSlice`: gestiona las "traducciones" de los códigos SIGPAC, que por sí mismos son solo identificadores numéricos o alfanuméricos. Al precargarlos, se garantiza que cualquier vista de detalle pueda mostrar información comprensible sin necesidad de solicitarla repetidamente.

En conjunto, esta estrategia asegura que el usuario arranque siempre con la información clave disponible, mejora el rendimiento al reducir llamadas redundantes al servidor y mantiene un estado global coherente que simplifica el trabajo de los distintos módulos de la aplicación.

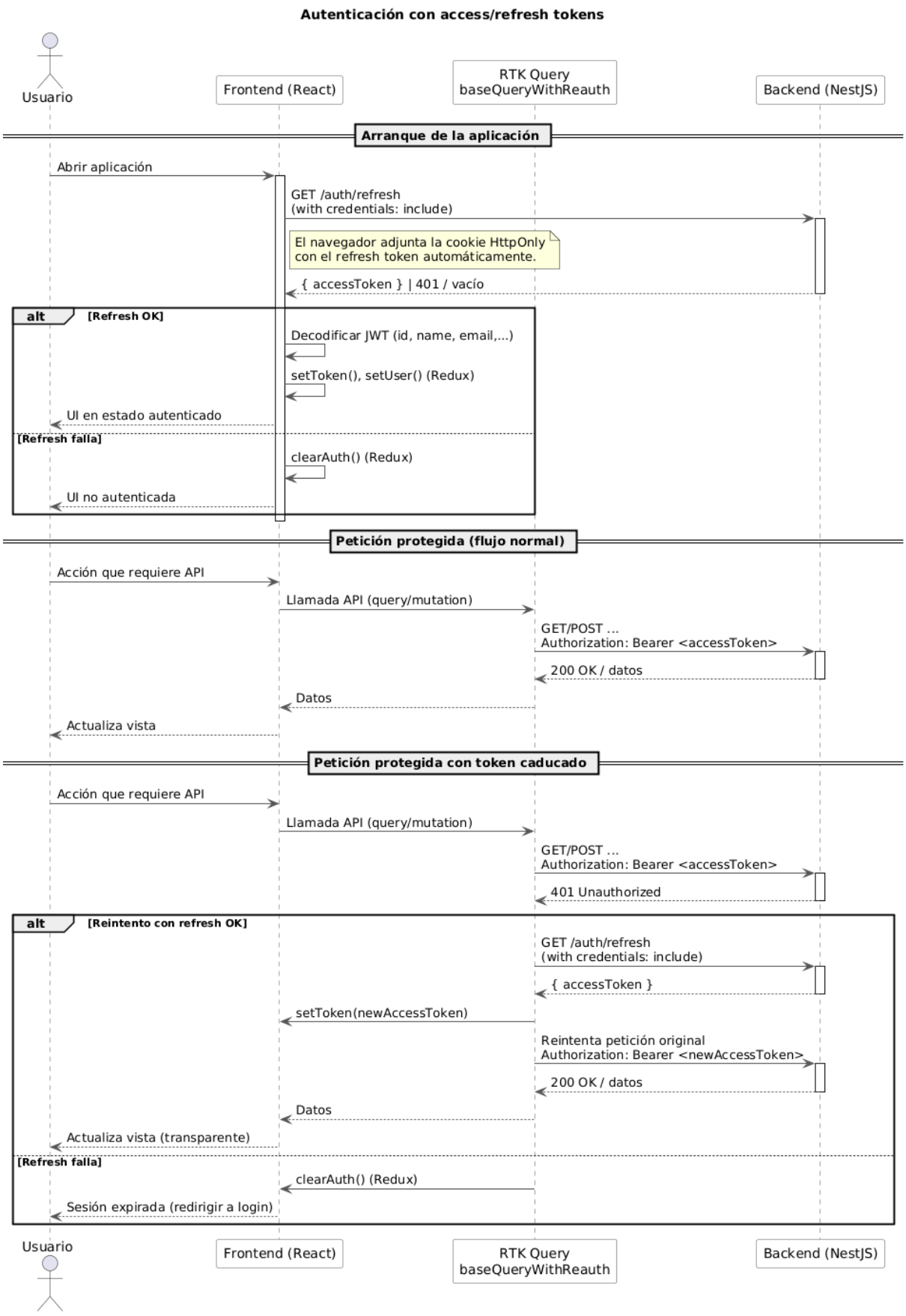

Figura 6.4: Diagrama de secuencia — Autenticación con *refresh tokens*

```tsx
export const useAuthInit = () => {
  const dispatch = useDispatch();
  const token = useSelector((state: RootState) => state.auth.token);

  const [authChecked, setAuthChecked] = useState(false);

  const {
    isLoading: dataLoading,
    isError: dataError
  } = useLoadAppData();

  // Refrescar sesión al inicio
  useEffect(() => {
    const refresh = async () => {
      try {
        const res = await fetch(API_URL + '/auth/refresh', {
          credentials: 'include',
        });
        const data = await res.json();

        if (data.accessToken) {
          const decoded = jwtDecode<TokenPayload>(data.accessToken);
          dispatch(setToken(data.accessToken));
          dispatch(setUser({
            id: decoded.id,
            name: decoded.name,
            email: decoded.email,
            preferredProvincia: decoded.preferredProvincia,
            preferredMunicipio: decoded.preferredMunicipio
          }));
        } else {
          dispatch(clearAuth());
        }
      } catch (err) {
        dispatch(clearAuth());
      } finally {
        setAuthChecked(true);
      }
    };

    refresh();
  }, [dispatch]);

  const isAuthenticated = !!token;

  const isLoading = !authChecked || (isAuthenticated && dataLoading);
  const isError =  isAuthenticated && dataError;
  const isUnauthenticated = authChecked && !isAuthenticated;

  return {
    isLoading,
    isError,
    isAuthenticated,
    isUnauthenticated,
  };
};
```

Figura 6.5: Implementación Frontend — `useAuthInit`

```tsx
export const useLoadAppData = () => {
  const dispatch = useDispatch();
  const provinciasMunicipios = useSelector((state: RootState) => state.sigpacCodes.provinciasMunicipios);

  const { data: parcels, isSuccess: parcelsReady, isLoading: parcelsLoading, isError: parcelsError } = useGetParcelsSummaryQuery();
  const { data: provincias, isSuccess: provinciasReady, isLoading: provinciasLoading, isError: provinciasError } = useGetSigpacCodes_ProvinciasQuery();
  const { data: usos, isSuccess: usosReady, isLoading: usosLoading, isError: usosError } = useGetSigpacCodes_UsosQuery();
  const { data: aprovechamientos, isSuccess: aprovechamientosReady, isLoading: aprovechamientosLoading, isError: aprovechamientosError } = useGetSigpacCodes_AprovechamientosQuery();
  const { data: incidencias, isSuccess: incidenciasReady, isLoading: incidenciasLoading, isError: incidenciasError } = useGetSigpacCodes_IncidenciasQuery();

  const [triggerGetMunicipios] = useLazyGetSigpacCodes_MunicipiosQuery();
  const [municipiosLoading, setMunicipiosLoading] = useState(true);
```

Figura 6.6: Implementación Frontend — `useLoadAppData` (1)

```tsx
  useEffect(() => {
    const loadMunicipios = async () => {
      if (parcelsReady && parcels && provinciasReady && provincias?.codigos) {
        dispatch(setParcels(parcels));

        const provinciasUnicas = [...new Set(parcels.map(p => p.sigpacData.provincia))];

        for (const codigoProvincia of provinciasUnicas) {
          const yaCargada = provinciasMunicipios.find(p => p.provincia.codigo === codigoProvincia && p.municipios.length > 0);
          if (yaCargada) continue;

          try {
            const { data } = await triggerGetMunicipios({ provincia: codigoProvincia }).unwrap();
            if (data?.codigos) {
              dispatch(addMunicipios({
                provinciaCodigo: codigoProvincia,
                municipios: data.codigos,
              }));
            }
          } catch (error) {
            console.warn(`No se pudieron cargar los municipios de la provincia ${codigoProvincia}`, error);
          }
        }
      }

      setMunicipiosLoading(false);
    };

    if (provinciasReady && provincias?.codigos) dispatch(setProvincias(provincias.codigos));
    if (usosReady && usos?.codigos) dispatch(setUsos(usos.codigos));
    if (aprovechamientosReady && aprovechamientos?.codigos) dispatch(setAprovechamientos(aprovechamientos.codigos));
    if (incidenciasReady && incidencias?.codigos) dispatch(setIncidencias(incidencias.codigos));

    loadMunicipios();
  }, [
    dispatch,
    provinciasReady, provincias,
    usosReady, usos,
    aprovechamientosReady, aprovechamientos,
    incidenciasReady, incidencias,
    parcelsReady, parcels,
    provinciasMunicipios,
    triggerGetMunicipios,
  ]);

  const isLoading =
    provinciasLoading || usosLoading || aprovechamientosLoading ||
    incidenciasLoading || parcelsLoading || municipiosLoading;

  const isError =
    provinciasError || usosError || aprovechamientosError ||
    incidenciasError || parcelsError;

  return {
    isLoading,
    isError,
  };
};
```

Figura 6.7: Implementación Frontend — `useLoadAppData` (2)

6.1.4. Enrutamiento y control de acceso

El sistema de navegación se ha construido sobre `React Router`, lo que permite estructurar la aplicación en distintas páginas y controlar el acceso a ellas en función del estado de autenticación del usuario.

El archivo principal `App.tsx` actúa como punto de entrada:

- Mientras la aplicación inicializa la sesión (`isLoading`) o detecta errores de conexión (`isError`), se muestran pantallas superpuestas mediante el componente `InfoOverlay`.
- Una vez resuelta la autenticación, el componente `Header` se renderiza únicamente si el usuario está identificado, manteniendo así la coherencia entre estado y navegación.
- El cuerpo principal de la aplicación se delega al componente `RouteStack`, que centraliza todas las rutas.

Rutas públicas y privadas En `RouteStack.tsx` (ver Figura 6.8) se organiza la navegación en dos grupos:

- **Rutas públicas restringidas**: aquellas a las que solo se puede acceder si el usuario no ha iniciado sesión, como la página de bienvenida o *landing*. Se encapsulan dentro de `PublicOnlyRoute`.
- **Rutas privadas**: se accede únicamente cuando el usuario está autenticado. Incluyen la *Home*, el visor de parcelas, la gestión de parcelas y el perfil. Estas rutas se encapsulan con `PrivateRoute`.

Mecanismos de control Los componentes `PrivateRoute` (ver Figura 6.9) y `PublicOnlyRoute` (ver Figura 6.10) son responsables de aplicar la lógica de control de acceso:

- `PrivateRoute`: comprueba si el estado de `authSlice` contiene un `token` y un `user` válidos. Si no es así, redirige automáticamente a la ruta `/landing`.
- `PublicOnlyRoute`: impide el acceso a páginas públicas cuando el usuario ya está autenticado, redirigiendo hacia la página principal (`/`).

Gestión de errores Cualquier ruta no definida en el sistema de navegación es capturada por la ruta comodín `*`, que muestra el componente `NotFound`. De este modo, se garantiza que la aplicación siempre ofrezca una respuesta controlada incluso en situaciones de navegación incorrecta.

Este esquema asegura una experiencia consistente: los usuarios autenticados acceden directamente a las funcionalidades principales, mientras que los no autenticados quedan restringidos a páginas de acceso público, manteniendo así la seguridad y la usabilidad.

6.1.5. Otras consideraciones

Además de los aspectos principales descritos, durante el desarrollo del frontend se han tenido en cuenta varias cuestiones técnicas relevantes:

```tsx
const RouteStack = () => {
  return (
    <Routes>
      {/* Rutas públicas accesibles solo si el usuario NO está autenticado */}
      <Route element={<PublicOnlyRoute />}>
        <Route path={routes.LANDING} element={<Landing />} />
      </Route>

      {/* Rutas privadas solo si el usuario está autenticado */}
      <Route element={<PrivateRoute />}>
        <Route path={routes.HOME} element={<Home />} />
        <Route path={routes.VISOR} element={<Visor />} />
        <Route path={routes.PARCELS_OVERVIEW} element={<ParcelsOverview />} />
        <Route path={routes.PARCEL_DETAIL} element={<ParcelDetail />} />
        <Route path={routes.PROFILE} element={<Profile />} />
      </Route>

      {/* 404 */}
      <Route path="*" element={<NotFound />} />
    </Routes>
  );
};

export default RouteStack;
```

Figura 6.8: Implementación Frontend — `RouteStack`

```tsx
const PrivateRoute = () => {
  const { token, user } = useSelector((state: RootState) => state.auth);
  const isAuthenticated = !!token && !!user;

  if (!isAuthenticated) {
    return <Navigate to="/landing" replace />;
  }

  return <Outlet />;
};

export default PrivateRoute;
```

Figura 6.9: Implementación Frontend — `PrivateRoute`

```tsx
const PublicOnlyRoute = () => {
  const { token, user } = useSelector((state: RootState) => state.auth);
  const isAuthenticated = !!token && !!user;

  if (isAuthenticated) {
    return <Navigate to="/" replace />;
  }

  return <Outlet />;
};

export default PublicOnlyRoute;
```

Figura 6.10: Implementación Frontend — `PublicOnlyRoute`

Estructuras de datos en la `store` La `store` de Redux no solo almacena el estado de autenticación, sino también entidades clave de la aplicación. Para mantener consistencia, se han definido interfaces TypeScript en `types/`, tales como `User`, `Parcel`, `Recinto` o `EconomicData`. Estas interfaces actúan como contratos que garantizan que todos los componentes y servicios trabajen con la misma forma de datos.

Validación de entradas Todos los formularios de la aplicación (registro, edición de perfil, creación de parcela, etc.) incluyen validaciones de los campos introducidos por el usuario. Mediante funciones auxiliares definidas dentro de la carpeta `utils/`, se validan aspectos como longitud mínima, formatos de correo electrónico o valores numéricos permitidos. Esto evita errores tempranos y asegura la integridad de los datos antes de enviarlos al backend.

Normalización de datos externos Los endpoints de SIGPAC y Catastro devuelven estructuras de datos heterogéneas, normalmente en formatos JSON con claves y convenciones poco consistentes. Para integrarlos en la aplicación, se aplican transformaciones que los adaptan a un formato estándar definido en los tipos internos. De este modo, cualquier componente puede trabajar con datos homogéneos sin preocuparse de la fuente original.

6.2. Backend

El backend se ha implementado con **NestJS**, siguiendo una arquitectura modular y capas bien definidas. A continuación se detallan los aspectos clave de la configuración y puesta en marcha del servidor, así como la estructura de módulos y los mecanismos transversales (seguridad, validación, CORS, ficheros estáticos y conexión a base de datos).

6.2.1. Arquitectura de NestJS

NestJS es un *framework* para Node.js orientado a la construcción de aplicaciones del lado del servidor siguiendo principios de la **arquitectura modular** e inspirándose en patrones como **MVC** y la **inyección de dependencias**. Su diseño recuerda a frameworks de otros lenguajes (como Angular en el frontend o Spring en Java), lo que facilita organizar proyectos de gran escala.

Los elementos principales de NestJS son:

- **Módulos**: son la unidad organizativa básica de una aplicación NestJS. Cada módulo agrupa controladores, servicios y entidades relacionados con una funcionalidad concreta. Por ejemplo, en este proyecto existen módulos como `UsersModule`, `ParcelsModule` o `AuthModule`. Todos se importan en el `AppModule`, que actúa como raíz de la aplicación.

- **Controladores**: definen los **endpoints HTTP** disponibles para cada módulo. Son responsables de recibir las peticiones del cliente (frontend), delegarlas a los servicios y devolver las respuestas. Por ejemplo, un `UsersController` podría manejar rutas como `/users/:id`.

- **Servicios**: contienen la **lógica de negocio** de la aplicación. Los controladores no implementan lógica compleja, sino que la derivan a los servicios. Gracias a la **inyección de dependencias**, un servicio puede ser fácilmente usado en distintos controladores o incluso en otros servicios.
- **Entidades y repositorios**: en proyectos que usan bases de datos (como PostgreSQL en este caso, mediante TypeORM), las entidades representan las tablas y sus columnas, mientras que los repositorios gestionan el acceso y manipulación de los datos.
- **Pipes, guards y filtros**: NestJS permite añadir **capas transversales**:
  - Los *pipes* transforman o validan datos de entrada (p.ej., el `ValidationPipe`).
  - Los *guards* controlan el acceso a rutas según la autenticación o roles (p.ej., `JwtAuthGuard`).
  - Los *filtros* gestionan excepciones y errores de forma centralizada.

En conjunto, NestJS ofrece una estructura clara y escalable que separa responsabilidades: los controladores se centran en la comunicación HTTP, los servicios en la lógica de negocio y los módulos en organizar cada área funcional.

6.2.2. Estructura de carpetas del backend

La organización de carpetas sigue la filosofía modular de NestJS, lo que facilita la escalabilidad y el mantenimiento del proyecto. Cada módulo agrupa sus controladores, servicios, entidades y DTOs de manera independiente, de forma que las responsabilidades están claramente separadas.

- **public/images/parcels**: almacena las imágenes y metadatos generados por los scripts Python, organizados por referencia catastral.
- **scripts**: contiene los scripts auxiliares en Python para la generación de imágenes y análisis mediante IA. Incluye también sus dependencias (`requirements.txt`).
- **src**: carpeta principal del código del backend, organizada en módulos:
  - **analyze-parcel**: módulo para ejecutar el análisis de recintos con IA. Incluye su servicio, controlador y la entidad `ai-data.entity.ts`.
  - **auth**: módulo de autenticación, con DTOs, guards, estrategias y controladores relacionados con login y seguridad.
  - **economic-data**: módulo que gestiona la información económica asociada a las parcelas.
  - **generate-image**: módulo encargado de generar recortes de parcelas a partir de su referencia catastral.
  - **parcels**: módulo principal para la gestión de parcelas. Incluye varias entidades (`parcel`, `recinto`, `sigpac_parcel`, `parcel_image`) y sus controladores y servicios asociados.
  - **users**: módulo de gestión de usuarios. Contiene la entidad `user.entity.ts`, junto con el controlador y el servicio de usuarios.

- Archivos principales:
  - `main.ts`: punto de arranque del servidor NestJS.
  - `app.module.ts`: módulo raíz que importa y coordina los demás módulos.

El archivo de configuración `.env`

En la raíz del proyecto se encuentra el archivo `.env`, que centraliza todas las variables de entorno necesarias para la ejecución del backend. Esto permite separar la configuración sensible (como contraseñas, claves secretas o API keys) del código fuente, facilitando su modificación sin necesidad de alterar la lógica del sistema.

El archivo contiene, entre otros, los siguientes parámetros:

- **Puerto del servidor**:

```
PORT=3000
```

- **URLs del frontend**: se utilizan para configurar CORS según el entorno de despliegue.

```
FRONTEND_URL_DEV=http://localhost:5173
FRONTEND_URL_PROD=https://mi-frontend.com
```

- **Conexión a la base de datos PostgreSQL**:

```
DB_HOST=localhost
DB_PORT=5432
DB_USER=admin
DB_PASS=admin
DB_NAME=parcels_db
```

- **JWT para autenticación**: incluye los secretos utilizados para firmar y verificar los *tokens*, así como su tiempo de expiración.

```
JWT_ACCESS_SECRET=secreto_seguro
JWT_REFRESH_SECRET=otro_secreto_seguro
JWT_ACCESS_EXPIRATION=15m
JWT_REFRESH_EXPIRATION=7d
```

- **APIs externas**: por ejemplo, la clave de Google Maps.

```
GOOGLE_API_KEY=google_api_key
```

6.2.3. Arranque del servidor (`main.ts`)

El código se expone en la Figura 6.11. El servidor se crea con `NestFactory.create(AppModule)` y se aplican configuraciones globales:

- **Validación y transformación** con `ValidationPipe`, que comprueba que los datos de entrada sean correctos y convierte automáticamente los tipos.
- **Cookies** con `cookie-parser`, necesarias para leer la cookie *HttpOnly* donde viaja el *refresh token*.
- **Prefijo global** `/api` para todas las rutas.
- **Archivos estáticos** servidos desde `/images` (carpeta `public/images`), donde se encuentran los recortes de imágenes satelitales y máscaras generadas por el modelo de IA.
- **CORS** habilitado para permitir que el frontend (Vite) envíe/reciba cookies entre orígenes.
- **Puerto** tomado de `.env` (`PORT`) o por defecto `3000`.

6.2.4. Módulo raíz y estructura (`AppModule`)

El `AppModule` (ver Figura 6.12) actúa como el punto de entrada principal del backend, donde se configuran y se ponen en marcha todas las piezas del sistema. En este módulo se indica qué otros módulos forman parte de la aplicación y cómo deben funcionar en conjunto. De forma resumida, el `AppModule` incluye:

- **ConfigModule**: permite leer las variables de configuración almacenadas en el archivo `.env`, como credenciales de la base de datos o el puerto del servidor.
- **TypeOrmModule.forRoot**: establece la conexión con la base de datos **PostgreSQL** usando los parámetros (`host`, `puerto`, `usuario`, `contraseña`, `nombre de la base de datos`). Además, se activa la opción de cargar automáticamente las entidades. Durante el desarrollo se ha usado `synchronize=true`, que crea o actualiza las tablas de forma automática. Sin embargo, en producción se debe desactivar para evitar riesgos de pérdida de datos.
- **ServeStaticModule**: sirve archivos estáticos (imágenes) desde la carpeta `public/images`, accesibles a través de la ruta `/images`.
- **Módulos de dominio**: agrupan funcionalidades específicas de la aplicación, como:
  - `AuthModule`: autenticación y seguridad.
  - `UsersModule`: gestión de usuarios.
  - `ParcelsModule`: gestión de parcelas.
  - `EconomicDataModule`: gestión de datos económicos.
  - `GenerateImageModule`: generación de imágenes de parcelas (recortes).
  - `AnalyzeParcelModule`: análisis de recintos con técnicas de IA.

```ts
async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  app.useGlobalPipes(
    new ValidationPipe({
      whitelist: true,
      transform: true,
    }),
  );

  app.use(cookieParser());
  app.setGlobalPrefix('api'); // prefijo para todas las rutas
  app.use('/images', express.static(path.join(__dirname, '..', 'public/images')));

  const isProd = process.env.NODE_ENV === 'production';

  app.enableCors({
    origin: isProd ? process.env.FRONTEND_URL_PROD : process.env.FRONTEND_URL_DEV,
    credentials: true,
  });

  const port = process.env.PORT ?? 3000;
  await app.listen(port);
  console.log(`Backend escuchando en http://localhost:${port}`);
}

bootstrap().catch((error) => {
  console.error('Error al iniciar el servidor:', error);
});
```

Figura 6.11: Implementación Backend — `main`

```ts
@Module({
  imports: [
    ServeStaticModule.forRoot({
      rootPath: join(__dirname, '..', 'public/images'), // ruta absoluta a /public/images
      serveRoot: '/images', // se accede por /images en la URL
    }),
    ConfigModule.forRoot({ isGlobal: true }), // para variables .env
    TypeOrmModule.forRoot({
      type: 'postgres',
      host: process.env.DB_HOST,
      port: parseInt(process.env.DB_PORT!),
      username: process.env.DB_USER,
      password: process.env.DB_PASS,
      database: process.env.DB_NAME,
      autoLoadEntities: true,
      synchronize: true, // en prod: false
    }),
    ParcelsModule,
    UsersModule,
    AuthModule,
    EconomicDataModule,
    GenerateImageModule,
    AnalyzeParcelModule
  ],
  controllers: [AppController],
  providers: [
    AppService,
    {
      provide: APP_GUARD,
      useClass: JwtAuthGuard,
    },
  ],
})
export class AppModule {}
```

Figura 6.12: Implementación Backend — `AppModule`

6.2.5. Autenticación con *refresh tokens* en el backend

Para implementar autenticación mediante *refresh tokens* en el backend, se definen el *guard* `JwtAuthGuard` (ver Figura 6.13), que actúa como filtro de seguridad en cada petición, y la estrategia `JwtStrategy` (ver Figura 6.14) para comprobar que el *token JWT* incluido en la cabecera `Authorization` sea válido. El funcionamiento es el siguiente:

- Cuando llega una petición, el guard consulta mediante un decorador (`@Public`) si esa ruta está marcada como pública. En tal caso, permite el acceso sin más comprobaciones.
- Si la ruta no es pública, el guard exige que la petición incluya un **token JWT** válido en la cabecera `Authorization: Bearer <token>`.
- La estrategia `JwtStrategy` se encarga de verificar la firma del token con la clave secreta del servidor (`JWT_ACCESS_SECRET`) y de rechazarlo si está caducado o manipulado.
- Si la validación es correcta, el payload del token se inyecta en el contexto de la petición, permitiendo a los controladores acceder directamente a la información del usuario autenticado.

De esta manera, el `JwtAuthGuard` asegura que sólo los usuarios autenticados puedan acceder a los **endpoints privados**, mientras que los públicos (por ejemplo, el login o el registro) se marcan explícitamente como accesibles sin autenticación.

```
@Injectable()
export class JwtAuthGuard extends AuthGuard('jwt') {
  constructor(private reflector: Reflector) {
    super();
  }

  canActivate(context: ExecutionContext) {
    const isPublic = this.reflector.getAllAndOverride<boolean>(IS_PUBLIC_KEY, [
      context.getHandler(),
      context.getClass(),
    ]);
    if (isPublic) return true;

    return super.canActivate(context);
  }
}
```

Figura 6.13: Implementación Backend — JWT Guard

6.2.6. Generación de recortes y análisis IA

Los recortes de imágenes geoespaciales y anásis mediante el modelo de Inteligencia Artificial se realiza mediante **scripts externos en Python**, ubicados en la carpeta `scripts`. Existen dos servicios principales que hacen uso de esta integración:

- **GenerateImageService**: correspondiente a la Figura 6.15. Invoca el script `generar-imagen-pro.py`, encargado de obtener imágenes recortadas de parcelas a partir de su referencia catastral (`refCat`) y otros códigos SIGPAC. El servicio genera un archivo PNG con la imagen de la parcela y un archivo JSON con metadatos (centro, zoom, límites, tamaño, etc.).

```
@Injectable()
export class JwtStrategy extends PassportStrategy(Strategy) {
  constructor() {
    super({
      jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
      ignoreExpiration: false,
      secretOrKey: process.env.JWT_ACCESS_SECRET,
    });
  }

  async validate(payload: any) {
    return { userId: payload.id, ...payload };
  }
}
```

Figura 6.14: Implementación Backend — JWT Strategy

- **AnalyzeParcelService**: correspondiente a la Figura 6.16. Ejecuta el script `process.py`, diseñado para analizar recintos de una parcela cuya imagen ha sido previamente generada. Este análisis produce imágenes correspondientes a las máscaras generadas y ficheros JSON con información estructurada como número de filas detectadas, longitudes, inclinación media o área total en metros cuadrados. El backend lee estos ficheros, transforma su contenido a objetos JSON y los devuelve al cliente.

Los resultados de los scripts se almacenan en la ruta:

`public/images/parcels/{refcat}`

- Imagen de la parcela completa: `{refcat}.png` y sus metadatos `{refcat}.json`.
- Imagen de cada recinto: `{refcat}_X.png`, donde `X` es el número de recinto, con su respectivo `.json`.
- Máscaras e información generadas por el modelo de IA: se guardan con el sufijo `_pp`, por ejemplo `{refcat}_X_pp.png` y `{refcat}_X_pp.json`.

6.2.7. Validación de datos con DTOs y Entities

Se ha implementado un mecanismo de validación de datos basado en **DTOs (Data Transfer Objects)** y en las **entidades de TypeORM (Entities)**. El objetivo es asegurar que la información que llega desde el cliente cumpla siempre con los requisitos esperados y evitar así errores o ataques.

- Los DTOs son clases que definen la forma exacta que deben tener los datos de entrada en cada operación. A través de decoradores de la librería `class-validator` se aplican reglas como:
  - `@IsString()`, `@IsInt()`, `@IsEmail()` para comprobar tipos básicos.
  - `@Length()`, `@Min()`, `@Max()` para validar rangos y longitudes.

```typescript
@Injectable()
export class GenerateImageService {
  async generate(refCat: string, pr: number, mu: number, ag: number, zo: number, po: number, pa: number ) {

    // Ruta absoluta al script Python
    const scriptPath = path.join(process.cwd(), 'scripts', 'generar-imagen-pro.py');

    // Directorio de salida (relativo a la raíz del proyecto)
    const outputDir = path.join(process.cwd(), 'public', 'images', 'parcels', refCat);
    const outputPng = path.join(outputDir, `${refCat}.png`);
    const outputJson = path.join(outputDir, `${refCat}.json`);

    // Solo genera si no existen los archivos
    if (!fsSync.existsSync(outputPng) || !fsSync.existsSync(outputJson)) {
      await execAsync(`python3 "${scriptPath}" --refcat "${refCat}" --output_dir "${outputDir}" --pr "${pr}" --mu "${mu}" --ag "${ag}" --zo "${zo}" --po "${po}" --pa "${pa}" `);
    }

    const metadataRaw = await fs.readFile(outputJson, 'utf-8');
    const metadata = JSON.parse(metadataRaw);

    return {
      center: metadata.center,
      zoom: metadata.zoom,
      image_size: metadata.image_size,
      margin: metadata.margin,
      bounds: metadata.bounds,
    };

  }
}
```

Figura 6.15: Implementación Backend — Servicio `GenerateImage`

```typescript
@Injectable()
export class AnalyzeParcelService {
  async analyze(refCat: string, recintos: number[]) {

    const processPath = path.join(process.cwd(), 'scripts', 'ai', 'process.py');

    const outputDir = path.join(process.cwd(), 'public', 'images', 'parcels', refCat);
    const recintosArgs = recintos.join(' ');

    const command = `python3 "${processPath}" --input_dir "${outputDir}" --output_dir "${outputDir}" --recintos ${recintosArgs}`;
    await execAsync(command);


    type RecintoMetadata = {
      recinto: number;
      numero_filas: number;
      longitudes_filas_m: number[];
      inclinacion_media_deg: number;
      area_total_m2: number;
    }

    const recintosMetadata: RecintoMetadata[] = [];

    for (const recintoId of recintos) {
      const recintoJsonPath = recintoId === -1
            ? path.join(outputDir, `${refCat}_pp.json`)
            : path.join(outputDir, `${refCat}_${recintoId}_pp.json`);

      try {
        const metadataRaw = await fs.readFile(recintoJsonPath, 'utf-8');
        const metadata = JSON.parse(metadataRaw);
        recintosMetadata.push({
          recinto: recintoId,
          ...metadata
        });
      } catch (err) {
        console.warn(`No se pudo leer el JSON de recinto ${recintoId}:`, err.message);
      }
    }

    return {
      data: recintosMetadata,
    };
  }
}
```

Figura 6.16: Implementación Backend — Servicio `AnalyzeParcel`

De este modo, si un cliente intenta enviar datos inválidos, NestJS bloquea la petición y devuelve un error de validación sin llegar a ejecutar la lógica de negocio.

- Las entidades son clases que representan las tablas de la base de datos. Con la librería `TypeORM` se definen mediante decoradores las columnas, claves primarias, relaciones y restricciones. Aunque su función principal es mapear los datos a PostgreSQL, las entities también sirven como **contrato estructural**, asegurando que la información almacenada en la base cumpla siempre con el modelo definido.

Como se puede ver en 6.11, se incorpora la *pipe* global `ValidationPipe`, configurada con las opciones:

- `whitelist=true`: elimina automáticamente cualquier campo no definido en el DTO, protegiendo frente a datos extra inesperados.
- `transform=true`: convierte automáticamente los tipos de datos de los parámetros (por ejemplo, de string a número).

6.2.8. Endpoints

En la Figura 6.17 se presenta un diagrama general de los distintos *endpoints* expuestos por el backend. Estos se encuentran organizados por módulos (autenticación, usuarios, parcelas, datos económicos, etc.) y cubren las operaciones de consulta, creación, actualización y eliminación de recursos.

Cada llamada a los *endpoints* se valida mediante el *token* JWT enviado por el cliente en la cabecera de la petición. De este modo, no se requiere pasar como parámetro el ID del usuario: el sistema identifica de manera inequívoca al usuario que realiza la petición y aplica los permisos correspondientes, garantizando tanto la seguridad como el aislamiento de los datos.

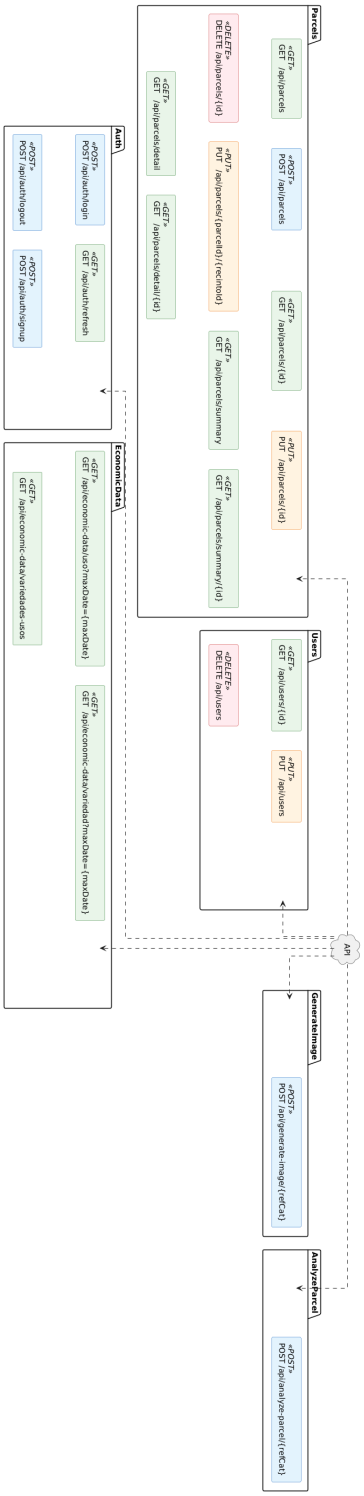

Figura 6.17: Implementación Backend — Endpoints

Capítulo 7

Pruebas y validación

En este capítulo se describen las pruebas realizadas para verificar que la aplicación cumple los requisitos definidos y funciona de forma robusta y segura. Se han aplicado dos enfoques complementarios: **pruebas de caja blanca**, centradas en la lógica interna del código, y **pruebas de caja negra**, orientadas al comportamiento externo del sistema.

7.1. Pruebas de caja blanca

Las pruebas de caja blanca consisten en verificar el correcto funcionamiento del sistema a partir de su estructura interna. Este tipo de pruebas se han llevado a cabo durante todo el desarrollo de la aplicación, tanto en el frontend como en el backend.

Entre las pruebas realizadas destacan:

- **Autenticación**: se verificó que el sistema de autenticación mediante *refresh tokens* funciona correctamente.
- **Enrutamiento**: se probó que los usuarios no autenticados solo pueden acceder a rutas definidas como públicas.
- **Manejo de errores**: se verificó que el sistema no colapsa frente a errores y notifica al usuario de forma clara.
- **Estados de carga**: se verificó que el sistema muestra estados de carga durante la obtención de datos externos.
- **Obtención de datos mediante APIs externas**: se validó el correcto funcionamiento de los servicios conectados al SIGPAC, Catastro y Open Meteo, asegurando que las respuestas se transforman al formato estándar usado internamente.
- **Persistencia de datos**: se comprobó que los datos persisten correctamente en la base de datos.
- **Consistencia de datos**: se verificó que en operaciones delicadas se mantiene la integridad referencial del sistema, respetando las validaciones y restricciones definidas en las entidades.
- **Seguridad**: se comprobó que el sistema está debidamente protegido contra ataques de inyección de código malicioso.

7.2. Pruebas de caja negra

Las pruebas de caja negra se centran en verificar el sistema desde el punto de vista externo, es decir, comprobando que las funcionalidades visibles para el usuario cumplen con los requisitos especificados. En este enfoque no se tiene en cuenta el código ni la estructura interna, sino las entradas, salidas y el comportamiento esperado.

Entre las pruebas realizadas destacan:

Tabla 7.1: PCN-01: Registro y login

PCN-01: Registro y login	
Objetivo	Validar que un usuario puede registrarse con datos válidos, iniciar sesión y acceder a la Home.
Precondiciones	No existe una cuenta previa con el correo de prueba. Backend disponible.
Datos de entrada	▪ Registro: `name, email, password, provincia, municipio` (válidos). ▪ Login: `email` y `password` recién creados.
Resultado esperado	▪ Registro: alta correcta y recepción de sesión activa. ▪ Login: redirección a `/` (Home) y cabecera visible.
Resultado obtenido	Conforme a lo esperado.

Tabla 7.2: PCN-02: Control de acceso

PCN-02: Control de acceso (rutas públicas/privadas)	
Objetivo	Verificar que sólo usuarios autenticados acceden a rutas privadas y que `/landing` no es accesible estando autenticado.
Precondiciones	Usuario creado. Estado autenticado/no autenticado según el caso.
Datos de entrada	▪ Sin sesión: navegar a `/`, `/parcels`, `/profile`. ▪ Con sesión: navegar a `/landing`.

PCN-02: Control de acceso (cont.)	
Resultado esperado	▪ Sin sesión: redirección a `/landing`. ▪ Con sesión: redirección automática a `/`.
Resultado obtenido	Conforme a lo esperado.

Tabla 7.3: PCN-03: Listado de parcelas

PCN-03: Listado de parcelas	
Objetivo	Comprobar que se muestran las parcelas del usuario.
Precondiciones	Usuario autenticado con varias parcelas registradas.
Datos de entrada	Navegar a `/parcels`.
Resultado esperado	▪ Se listan las parcelas con nombre, referencia y acciones. ▪ No hay llamadas redundantes; la vista se actualiza tras operaciones CRUD.
Resultado obtenido	Conforme a lo esperado.

Tabla 7.4: PCN-04: Alta/edición/eliminación de parcela

PCN-04: Alta/edición/eliminación de parcela (CRUD)	
Objetivo	Validar que el usuario puede crear, editar y borrar parcelas desde la UI.
Precondiciones	Usuario autenticado.
Datos de entrada	▪ Alta: nombre, descripción, `refCat` válidos. ▪ Edición: cambio de nombre/nota. ▪ Eliminación: confirmación de borrado.
Resultado esperado	▪ Alta/edición: feedback de éxito y listado actualizado. ▪ Eliminación: la parcela desaparece del listado.

PCN-04: Alta/edición/eliminación (cont.)	
Resultado obtenido	Conforme a lo esperado.

Tabla 7.5: PCN-05: Visor y recintos

PCN-05: Visor y recintos (SIGPAC/Catastro)	
Objetivo	Verificar que el visor carga la parcela seleccionada y sus recintos con datos asociados.
Precondiciones	Usuario autenticado. Conectividad con servicios externos.
Datos de entrada	Abrir una parcela en el visor; seleccionar recintos.
Resultado esperado	▪ Mapa centrado en la parcela. ▪ Recintos visibles y consultables (uso, superficie, etc.).
Resultado obtenido	Conforme a lo esperado.

Tabla 7.6: PCN-06: Datos económicos por recinto

PCN-06: Datos económicos por recinto	
Objetivo	Validar la introducción/actualización de datos económicos y su reflejo en la UI.
Precondiciones	Usuario autenticado, parcela con recintos.
Datos de entrada	▪ Variedad, producción, precio, costes, fecha (válidos).
Resultado esperado	▪ Guardado correcto y actualización inmediata de tablas/gráficas. ▪ Validación de formatos y rangos.
Resultado obtenido	Conforme a lo esperado.

Tabla 7.7: PCN-07: Meteorología

PCN-07: Meteorología (Open-Meteo)

PCN-07: Meteorología (cont.)	
Objetivo	Comprobar la consulta y visualización de previsión diaria y horaria.
Precondiciones	Conectividad con Open-Meteo. Usuario autenticado.
Datos de entrada	Coordenadas de la parcela/carácter geográfico seleccionado.
Resultado esperado	▪ Muestra datos meteorológicos. ▪ Manejo de errores de red con mensajes claros.
Resultado obtenido	Conforme a lo esperado.

Tabla 7.8: PCN-08: Eliminación de cuenta y limpieza de datos

PCN-08: Eliminación de cuenta y limpieza de datos	
Objetivo	Verificar que al eliminar la cuenta se eliminan datos asociados y se cierra la sesión.
Precondiciones	Usuario autenticado con parcelas y datos económicos.
Datos de entrada	Acción “Eliminar cuenta” y confirmación.
Resultado esperado	▪ Baja del usuario y borrado en cascada de parcelas/datos. ▪ Redirección a `/landing` y sesión finalizada.
Resultado obtenido	Conforme a lo esperado.

Capítulo 8

Manual de usuario

Este capítulo tiene como objetivo servir de guía práctica para cualquier persona que utilice la aplicación. Se describen de forma sencilla las diferentes pantallas y funcionalidades principales, acompañadas de capturas que ilustran su uso.

El manual está orientado a un usuario final, por lo que no se requiere ningún conocimiento técnico previo. Se explica cómo acceder al sistema, registrarse, gestionar las parcelas, consultar información detallada en el visor, revisar el perfil personal y cerrar la sesión de forma segura.

8.1. Landing page

La **Landing page** es la pantalla inicial de la aplicación, pensada para presentar las principales funcionalidades y animar al usuario a registrarse o iniciar sesión. En ella se destacan las ventajas de la plataforma, ejemplos visuales de uso y un acceso directo al formulario de registro.

En las Figuras 8.1, 8.2, 8.3 y 8.4 se muestran las diferentes secciones de la pantalla.

Figura 8.1: Manual de usuario — Landing page (1)

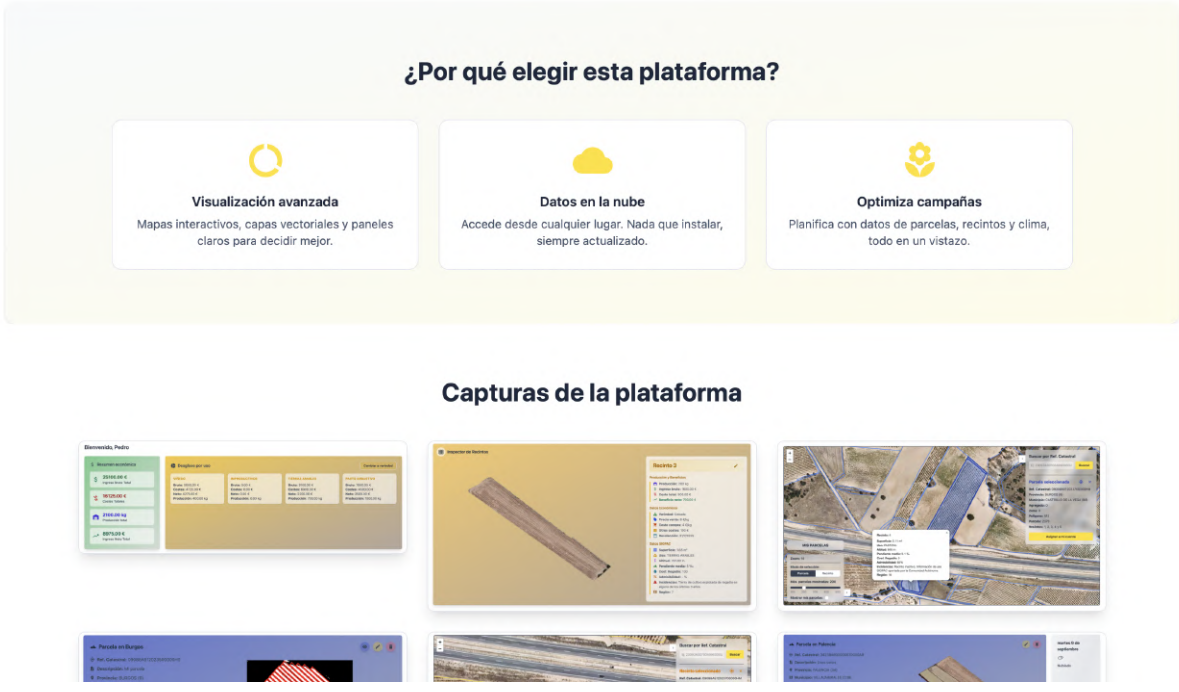

Figura 8.2: Manual de usuario — Landing page (2)

Figura 8.3: Manual de usuario — Landing page (3)

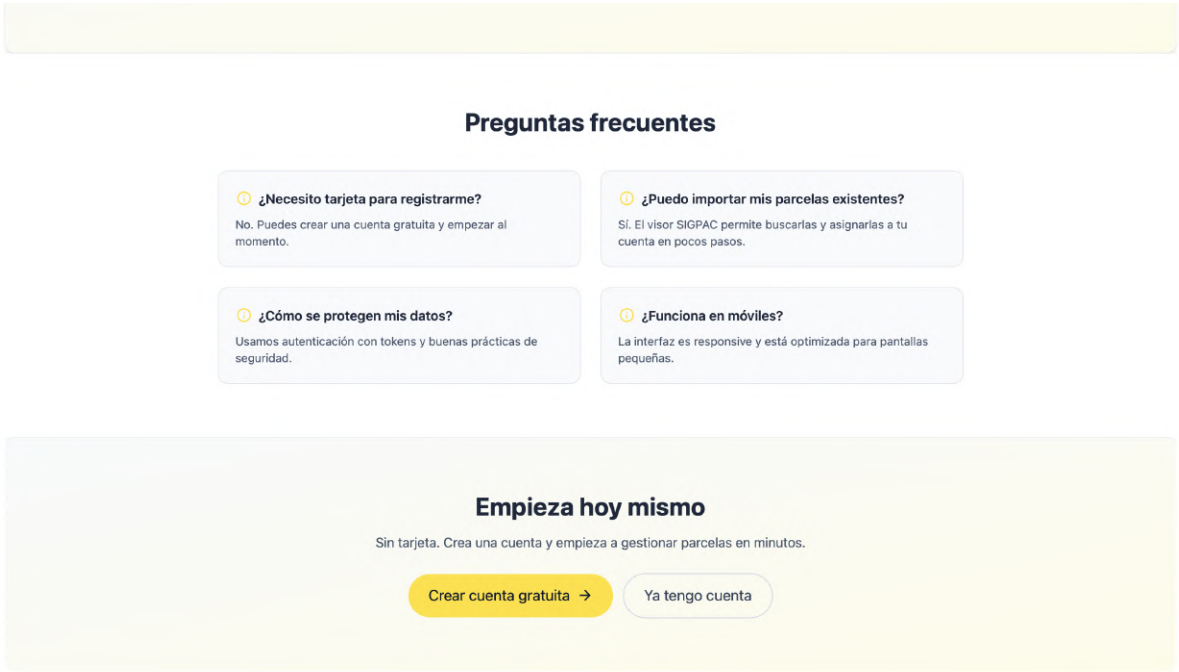

Figura 8.4: Manual de usuario — Landing page (4)

8.2. Registro y login

La plataforma cuenta con un sistema de autenticación basado en correo electrónico y contraseña. El acceso de los usuarios se realiza mediante dos modales: *Crear cuenta* y *Iniciar sesión* (Figuras 8.5, 8.6, 8.7 y 8.8).

- **Registro:** el modal de creación de cuenta solicita los siguientes datos:
  - Nombre del usuario.
  - Correo electrónico (único en el sistema).
  - Contraseña y confirmación de la misma.
  - Provincia y municipio de referencia.
  - Aceptación obligatoria del uso de cookies con fines de autenticación.

  Si algún campo obligatorio no se completa o se introduce un valor inválido, el sistema muestra mensajes de error específicos (Figura 8.6). En caso de que el correo ya esté registrado, se notifica al usuario para que utilice otro o inicie sesión (Figura 8.7).

- **Login:** el modal de inicio de sesión únicamente solicita correo electrónico y contraseña (Figura 8.8). Si las credenciales son correctas, el usuario accede directamente a la aplicación; en caso contrario, se informa del error.

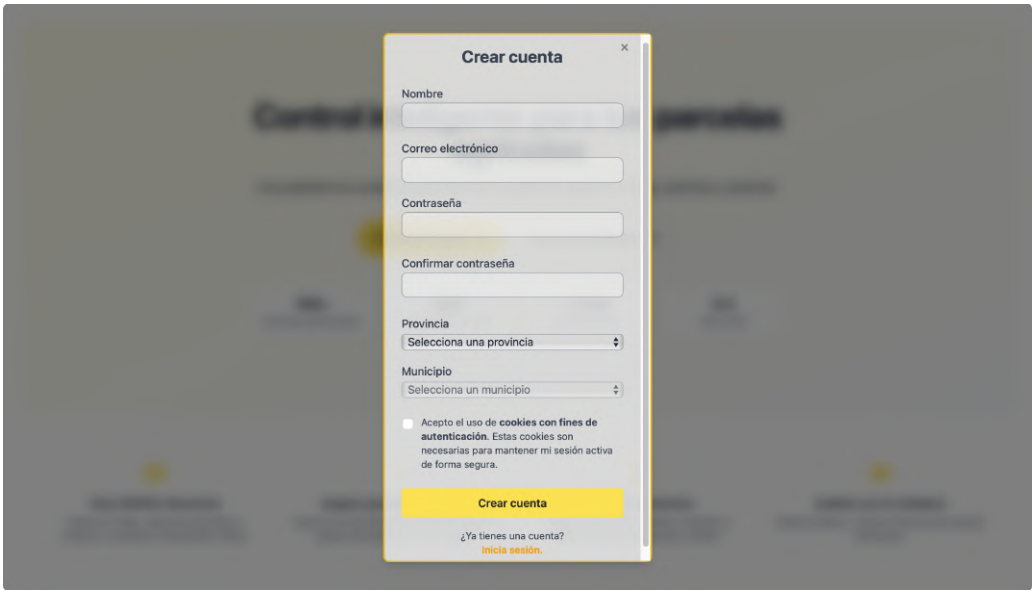

Figura 8.5: Manual de usuario — Registro de usuario (1)

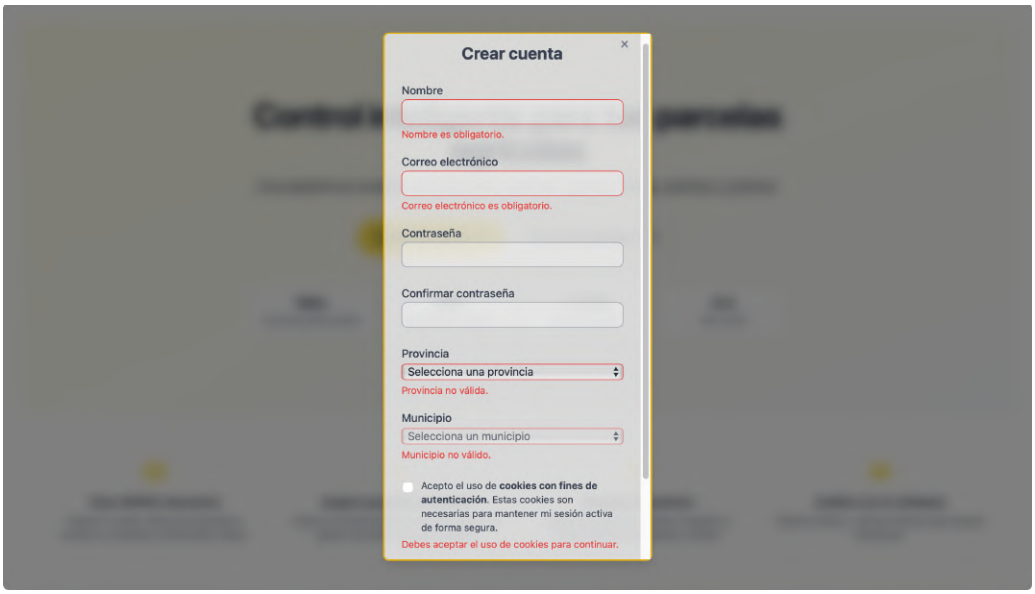

Figura 8.6: Manual de usuario — Registro de usuario (2)

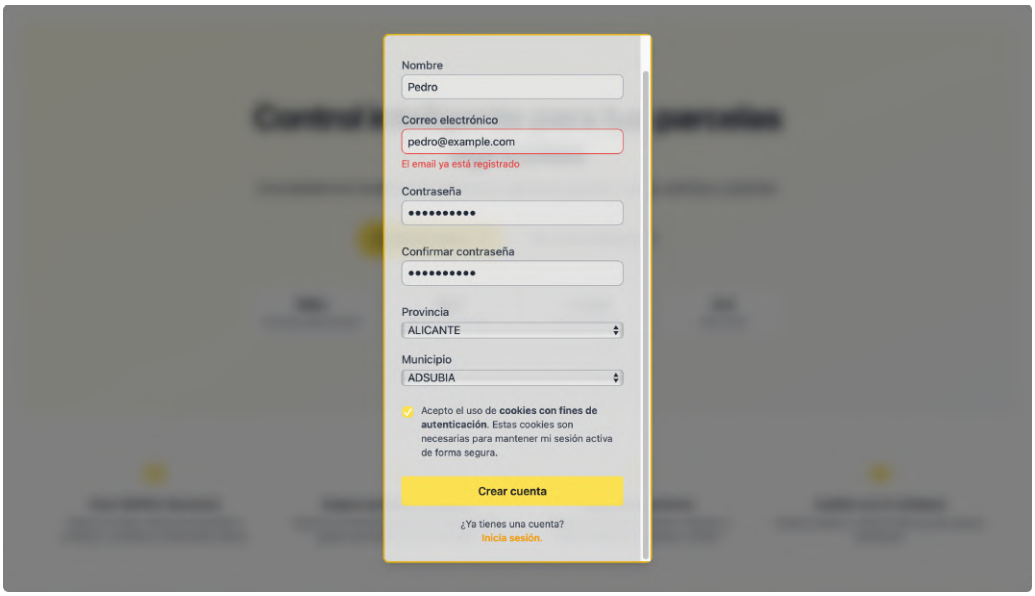

Figura 8.7: Manual de usuario — Registro de usuario (3o)

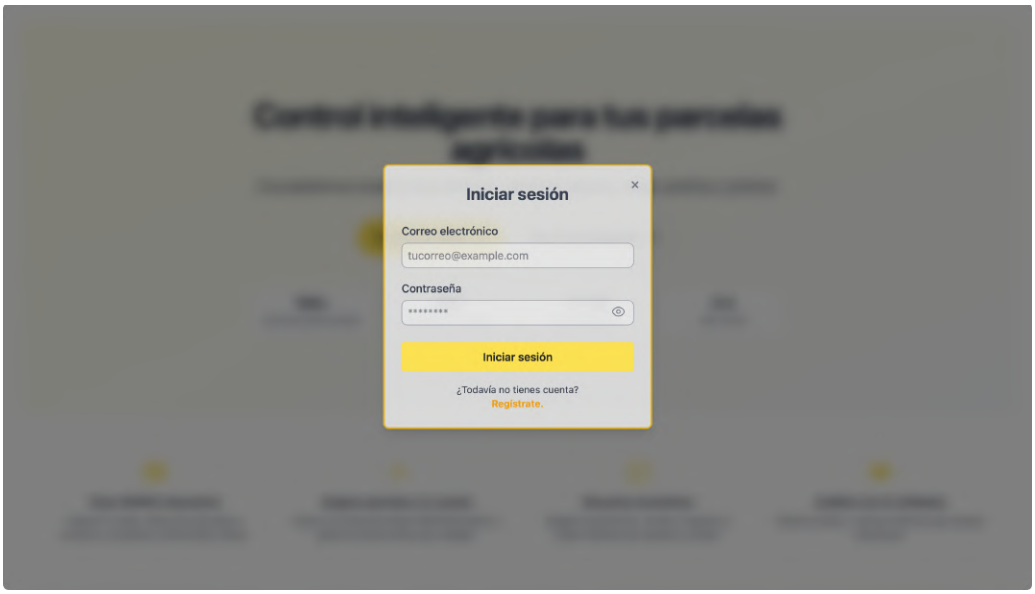

Figura 8.8: Manual de usuario — Login de usuario

8.3. Home

La página principal (*Home*) constituye el punto de acceso rápido a la información más relevante para el usuario. En ella se organizan distintos paneles o tarjetas que muestran datos en tiempo real y accesos directos a las funcionalidades más usadas de la aplicación.

Los elementos (paneles) que muestra la pantalla *Home* son los siguientes:

- **Resumen económico:** considera datos económicos de todas las parcelas del usuario.
- **Desglose por uso o variedad:** muestra datos económicos agrupados según uso SIGPAC o variedad de cultivo (definida por el usuario).
- **Parcelas vistas recientemente:** acceso rápido a las parcelas consultadas recientemente.
- **Tiempo en tu localidad:** muestra el tiempo actual y la previsión para los próximos días en la localidad por defecto del usuario.

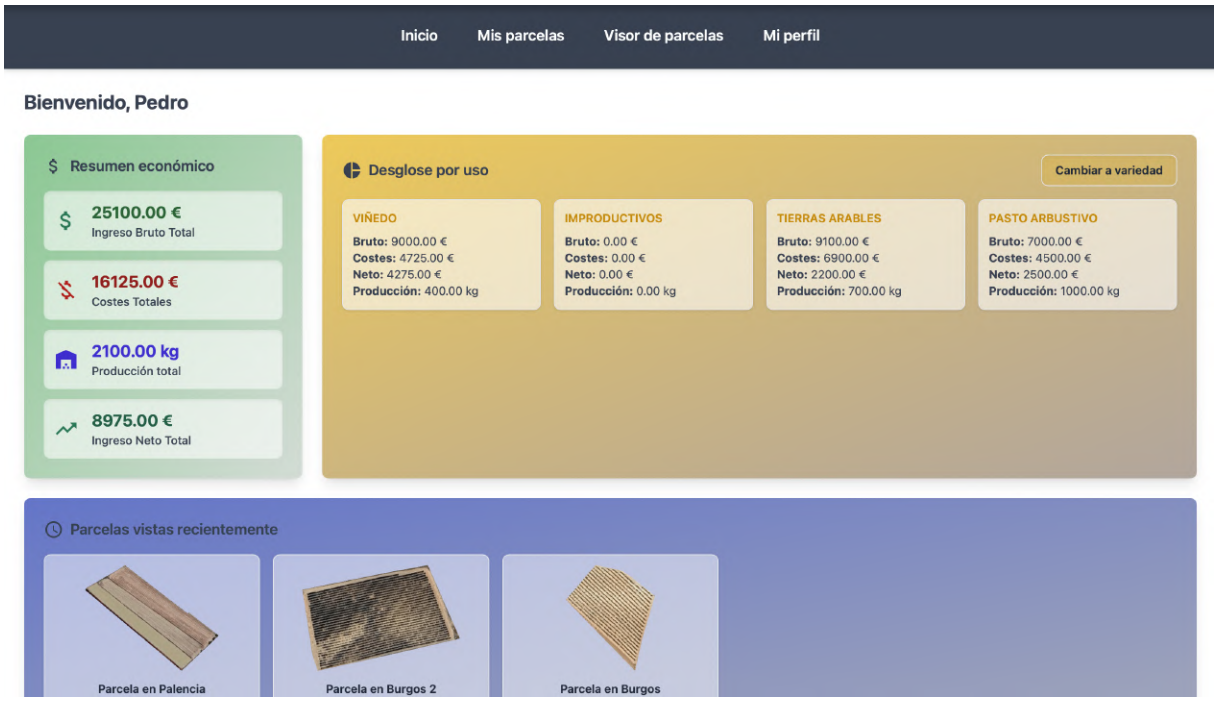

Figura 8.9: Manual de usuario — Home (1)

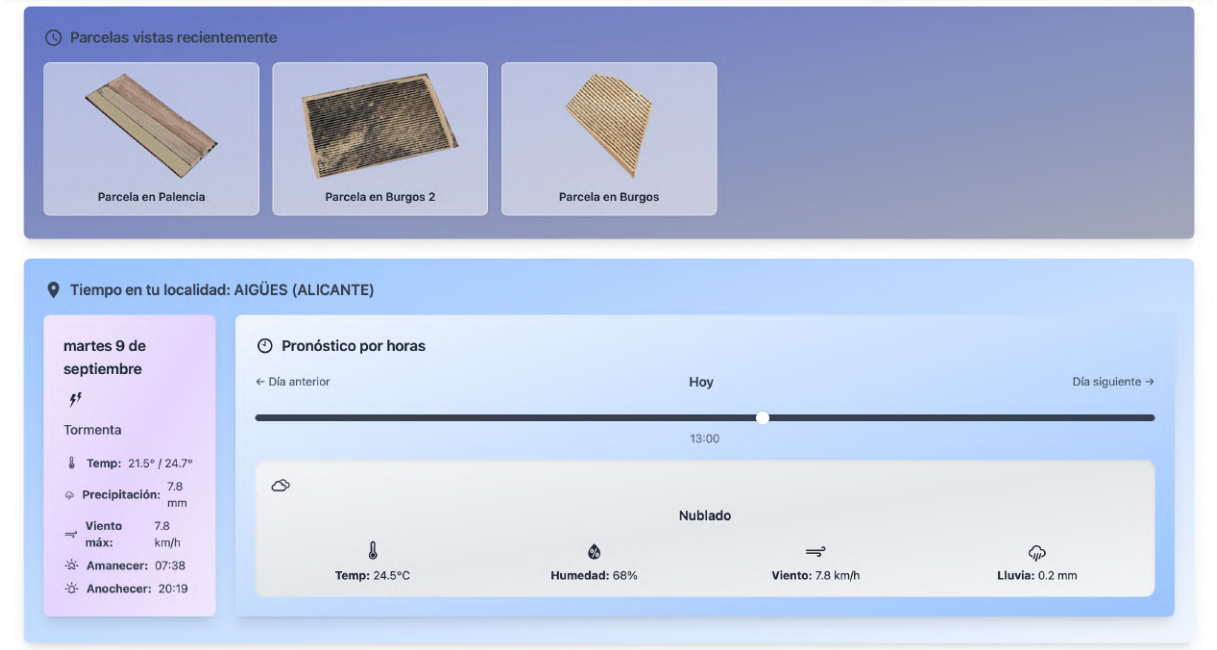

Figura 8.10: Manual de usuario — Home (2)

8.4. Mis parcelas

La sección *Mis parcelas* permite al usuario visualizar todas las parcelas que tiene registradas en la aplicación. En esta vista se listan las parcelas de las que es propietario, junto con información básica como su nombre, descripción y localización.

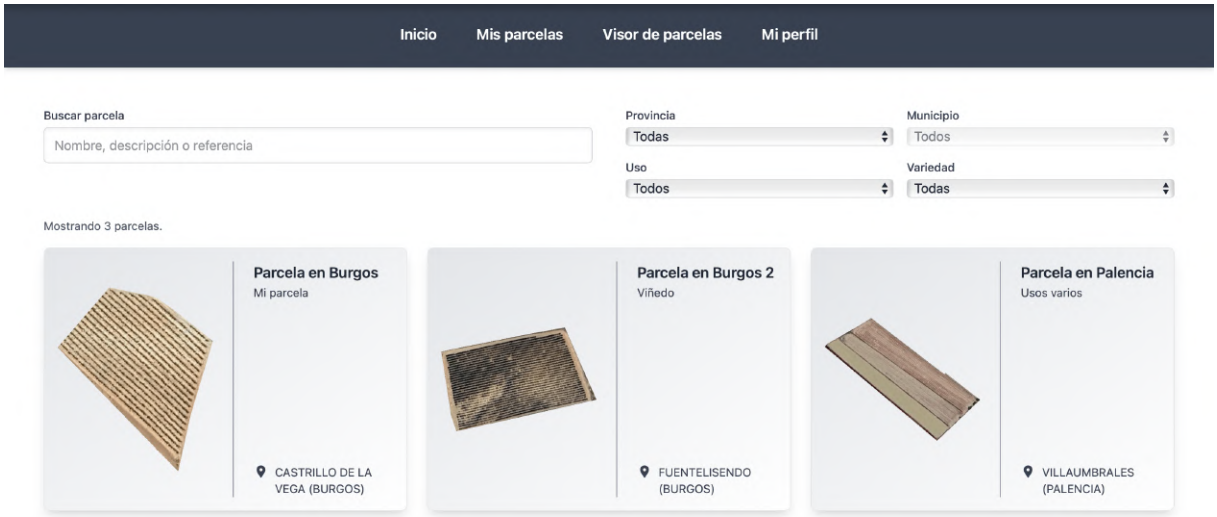

Figura 8.11: Manual de usuario — Mis parcelas (1)

En la parte superior se encuentra una sección de filtros, donde el usuario puede buscar una parcela por su nombre, descripción o referencia catastral (Figura 8.12); y/o filtrar según la localización de la parcela (Figura 8.13), su uso SIGPAC o su variedad.

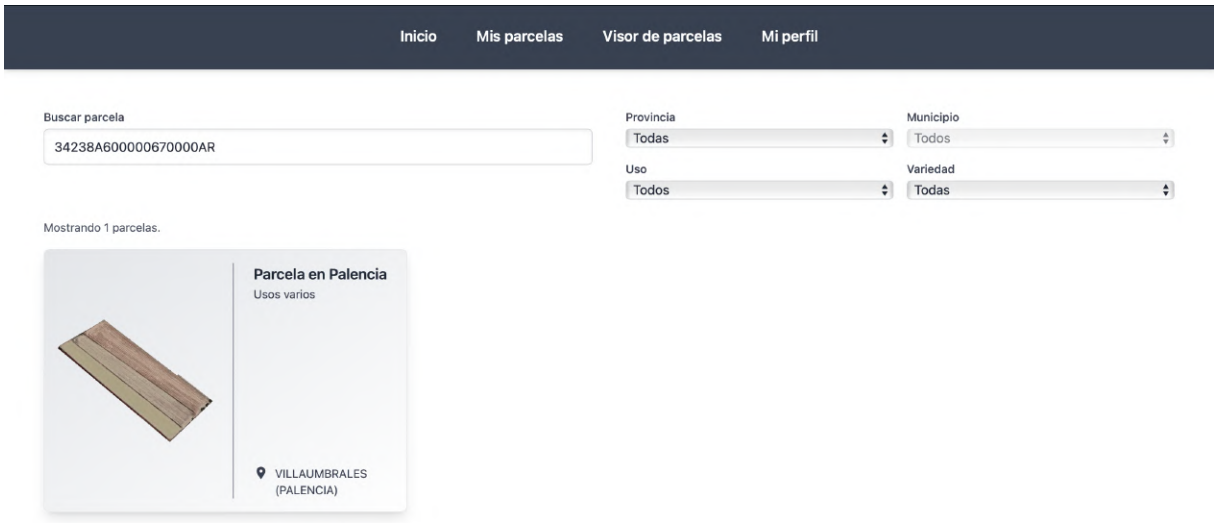

Figura 8.12: Manual de usuario — Mis parcelas (2)

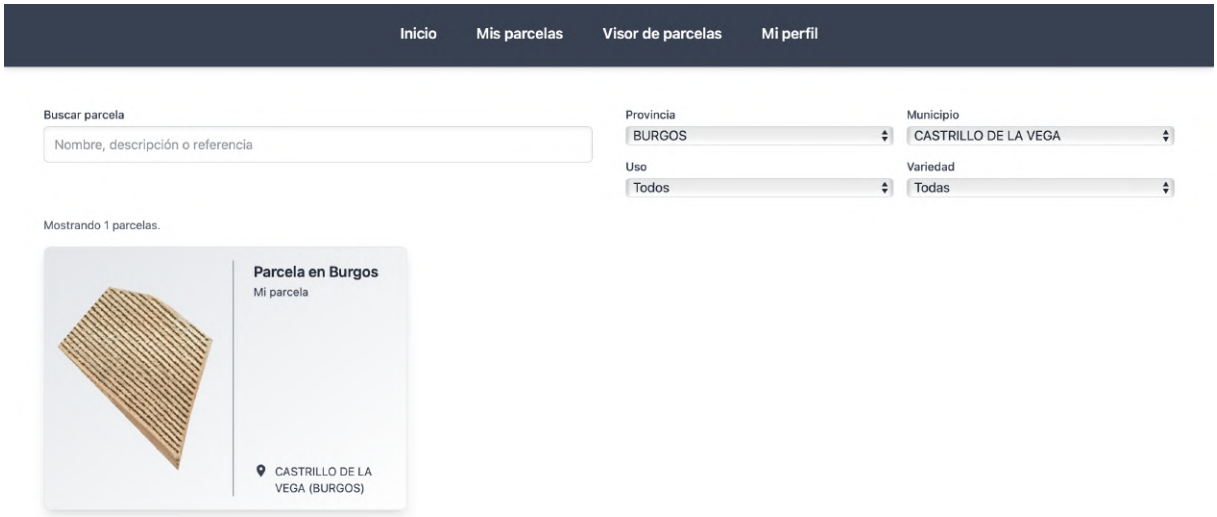

Figura 8.13: Manual de usuario — Mis parcelas (3)

Si el usuario no ha guardado ninguna parcela o los filtros aplicados no corresponden a ninguna parcela guardada, se muestra el mensaje *No se encontraron parcelas* (Figura 8.12).

Figura 8.14: Manual de usuario — Mis parcelas (4)

8.5. Detalle de parcela

En la vista de detalle se muestra toda la información disponible de una parcela seleccionada. La interfaz se divide en diferentes bloques (ver Figuras 8.15, 8.16 y 8.17):

- **Información de la parcela:** datos básicos obtenidos de SIGPAC y Catastro como referencia catastral, provincia, municipio, polígono, parcela y recintos asociados; así como los usos SIGPAC y variedades de los recintos que contiene. También se muestran los datos personalizables: nombre, descripción y notas. En la parte derecha del panel se encuentra la imagen de la parcela (recorte), y encima de esta botones de edición y borrado.
- **Resumen económico:** métricas calculadas a partir de los datos introducidos por el usuario: producción total, ingresos brutos, costes totales y beneficio neto.
- **Pronóstico meteorológico:** integrado con la API de *Open Meteo*, incluye:
  - Panel con la previsión del día de hoy (a la derecha del panel de información de la parcela).
  - Panel semanal con previsión de temperatura mínima y máxima, estado del cielo y precipitaciones.
  - Panel horario interactivo que permite visualizar el tiempo previsto para cada franja.
- **Inspector de recintos:** visualización gráfica de los recintos que conforman la parcela, diferenciados mediante contornos resaltados. Pulsando en cada recinto se muestra información específica del recinto.

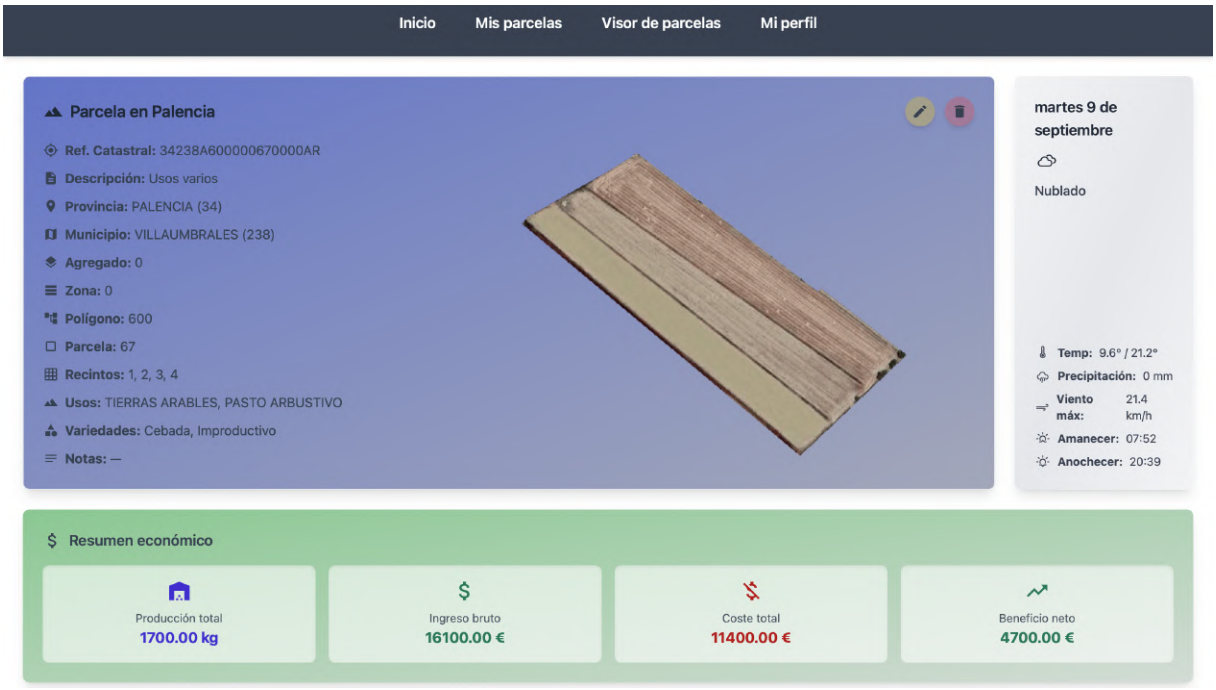

Figura 8.15: Manual de usuario — Detalle de parcela (1)

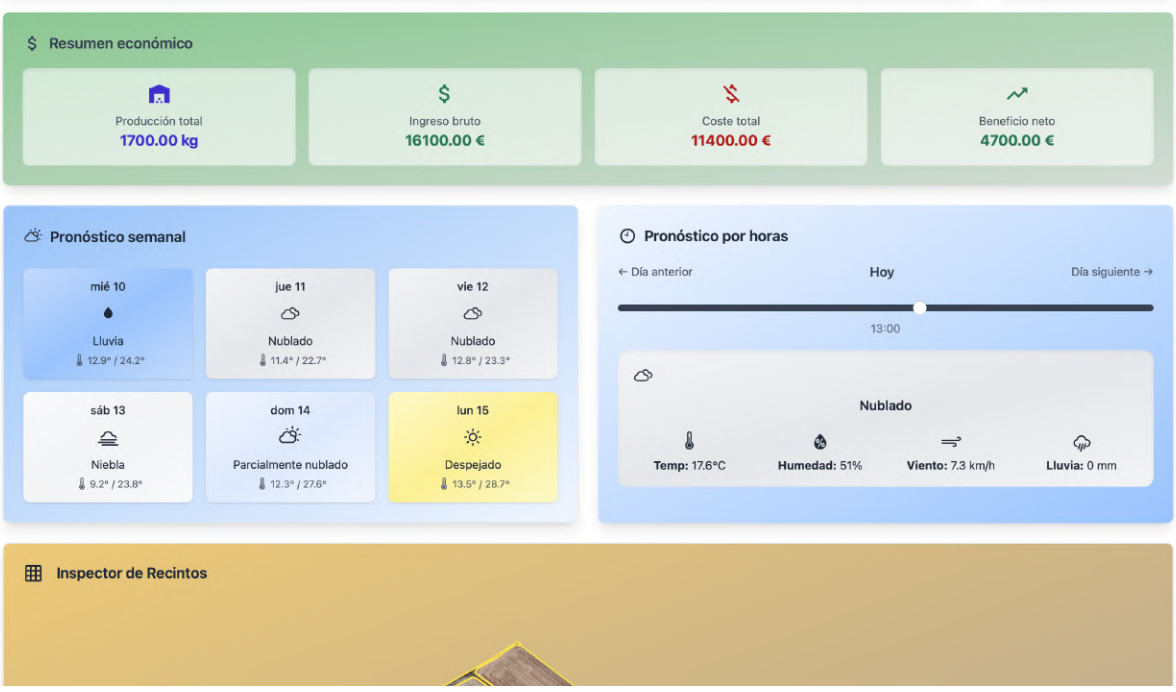

Figura 8.16: Manual de usuario — Detalle de parcela (2)

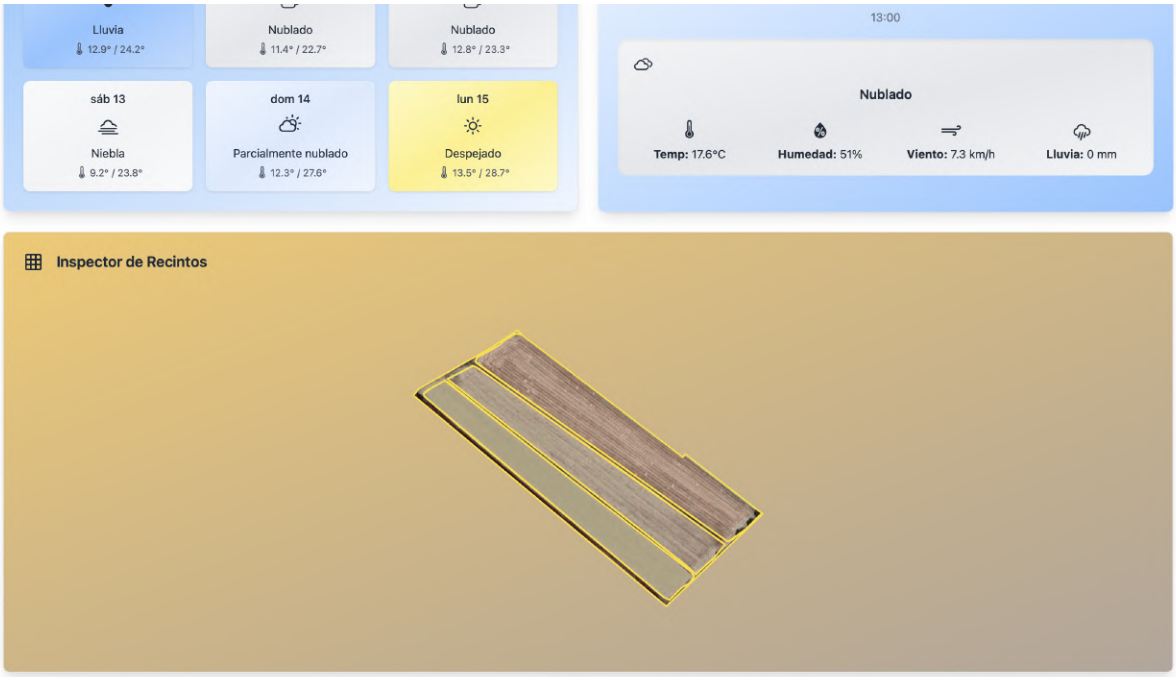

Figura 8.17: Manual de usuario — Detalle de parcela (3)

Edición y eliminación de la parcela

En el panel de información de la parcela, si se acciona el botón de edición (naranja con icono de lápiz), se activa el modo edición, permitiendo al usuario modificar el nombre, descripción y notas asociadas a la parcela (Figura 8.18).

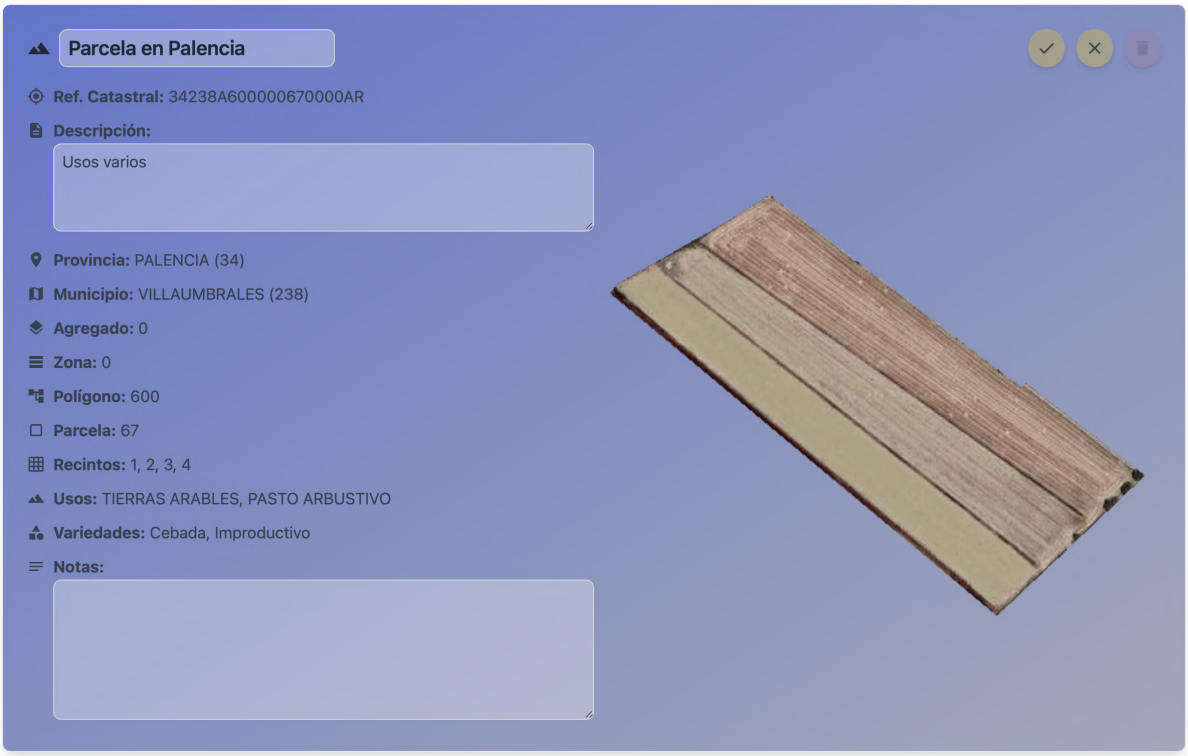

Figura 8.18: Manual de usuario — Edición de la parcela (1) (Detalle de parcela)

Modificando cualquier dato de la parcela y pulsando el botón de confirmar (naranja con icono de *tick*), se muestra un mensaje de éxito (Figura 8.19).

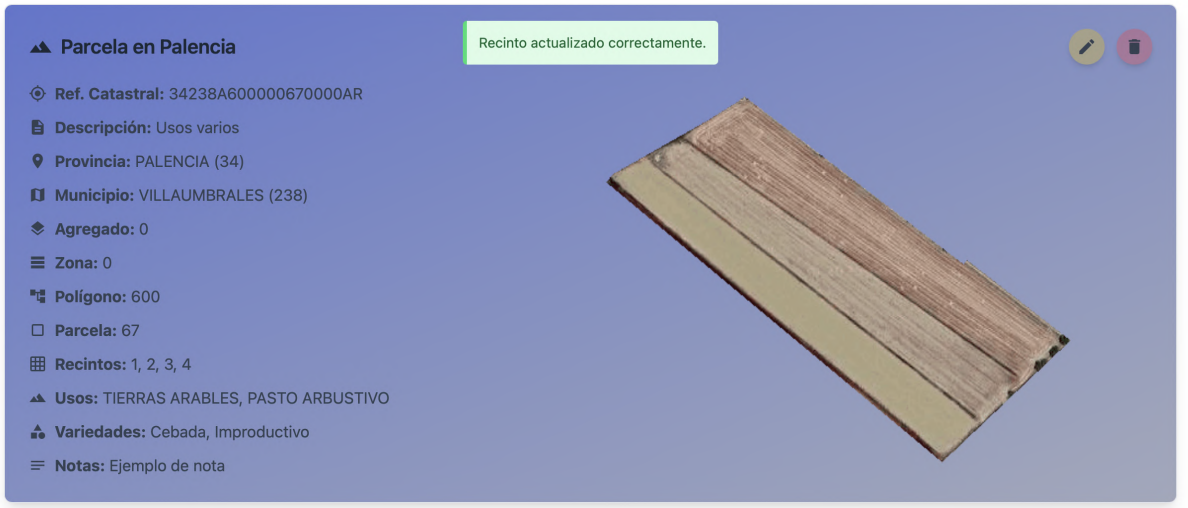

Figura 8.19: Manual de usuario — Edición de la parcela (2) (Detalle de parcela)

Si se acciona el botón de borrado (rojo con icono de papelera), aparecerá un modal para confirmar la eliminación de la parcela (Figura 8.20).

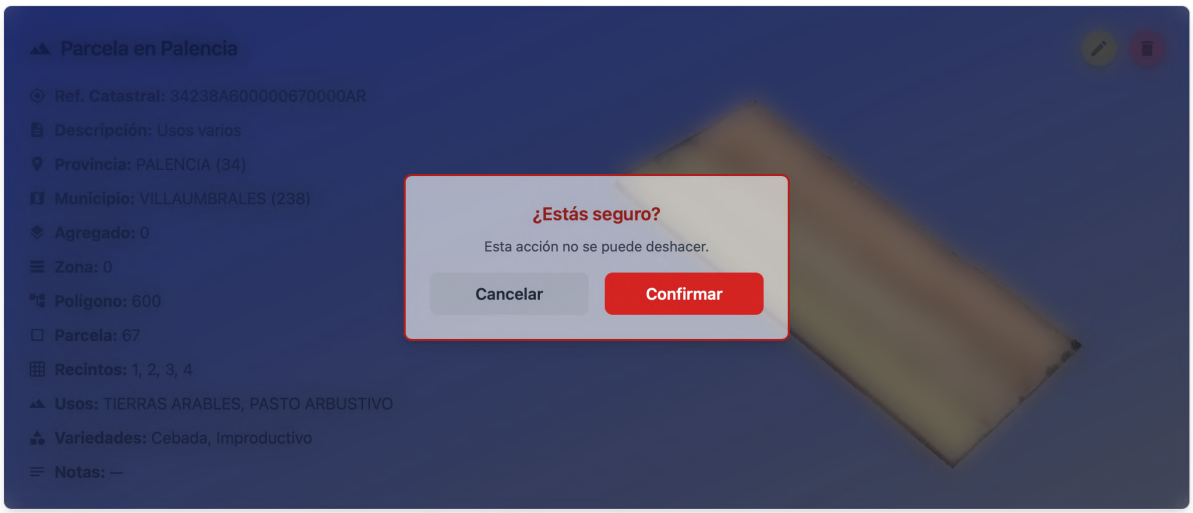

Figura 8.20: Manual de usuario — Eliminación de la parcela (Detalle de parcela)

Resumen económico

En el panel *Resumen económico*, pulsando en cualquiera de las cuatro métricas, se muestra un desglose por recintos (Figura 8.21). De nuevo, pulsando en cualquier recinto de este desglose, se hace scroll automático al Inspector de recintos, mostrando información detallada del recinto en cuestión.

Figura 8.21: Manual de usuario — Resumen económico (Detalle de parcela)

Inspector de recintos

En el panel *Inspector de recintos*, pulsando en un recinto de la parcela se muestra la información SIGPAC detallada del recinto, así como los datos económicos introducidos por el usuario y diferentes métricas relacionadas (Figura 8.22).

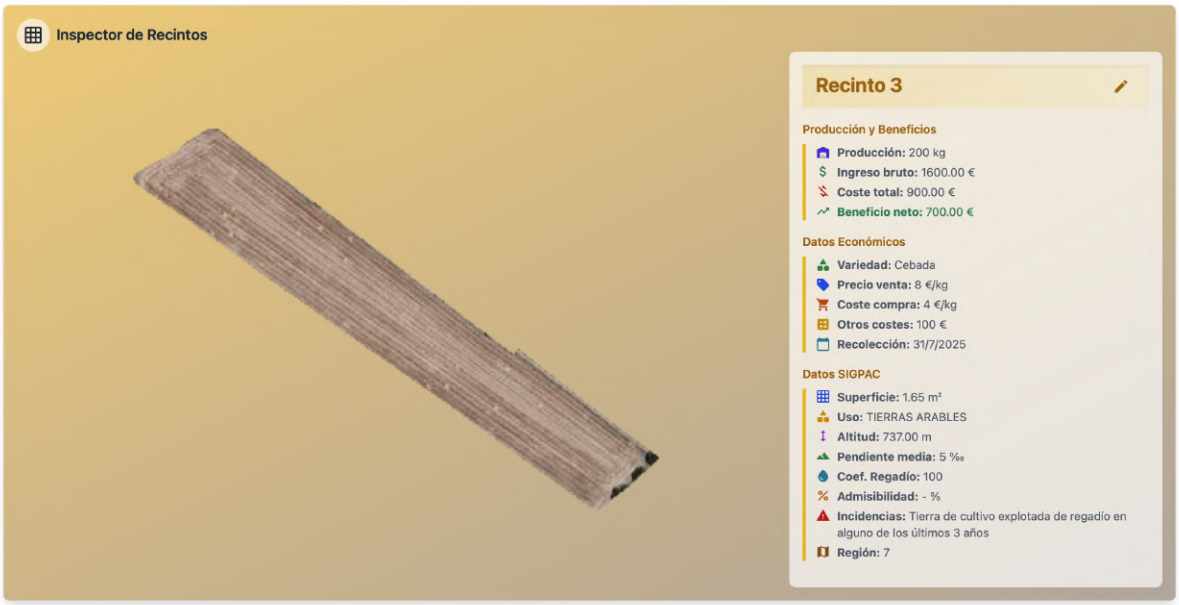

Figura 8.22: Manual de usuario — Inspector de recintos (1) (Detalle de parcela)

Pulsando el botón de edición (naranja con icono de lápiz) se activa el modo edición, permitiendo al usuario modificar los datos económicos asociados al recinto (Figura 8.23).

Figura 8.23: Manual de usuario — Inspector de recintos (2) (Detalle de parcela)

Viñedos

Si la parcela contiene recintos cuyo uso SIGPAC se clasifica como viñedo, tanto el panel de información de la parcela como el *Inspector de recintos* cambian.

En concreto, en el panel de información de la parcela aparece un nuevo botón (azul con icono de ojo), como se muestra en la Figura 8.24.

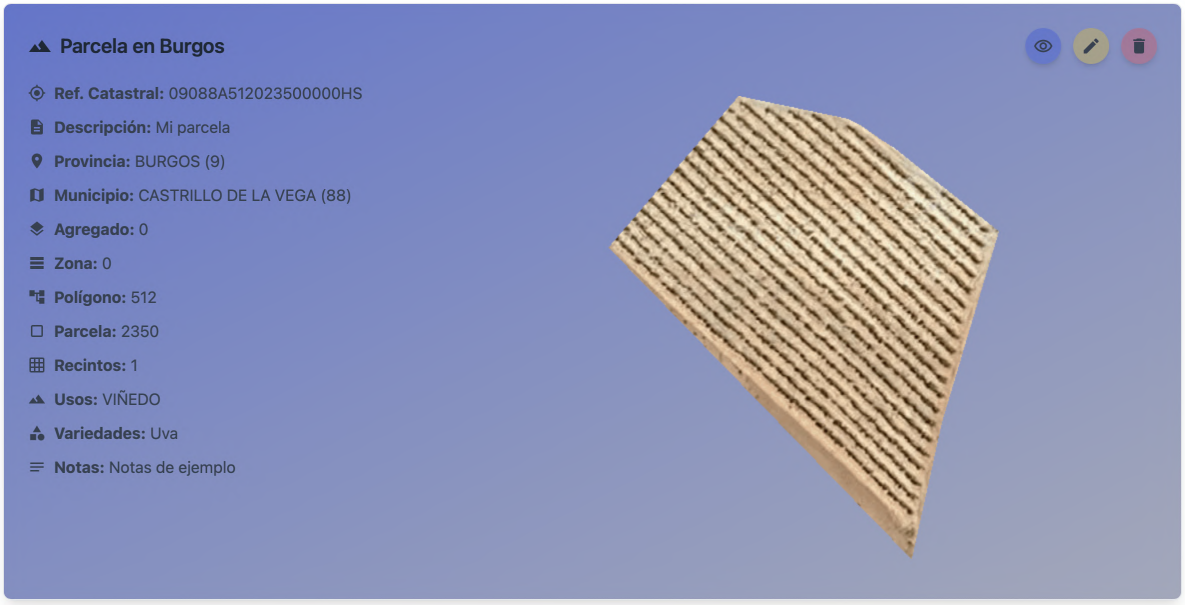

Figura 8.24: Manual de usuario — Viñedos (1) (Detalle de parcela)

Al pulsar este botón, la imagen de la parcela se reemplaza por la máscara generada por el modelo de IA (Figura 8.25).

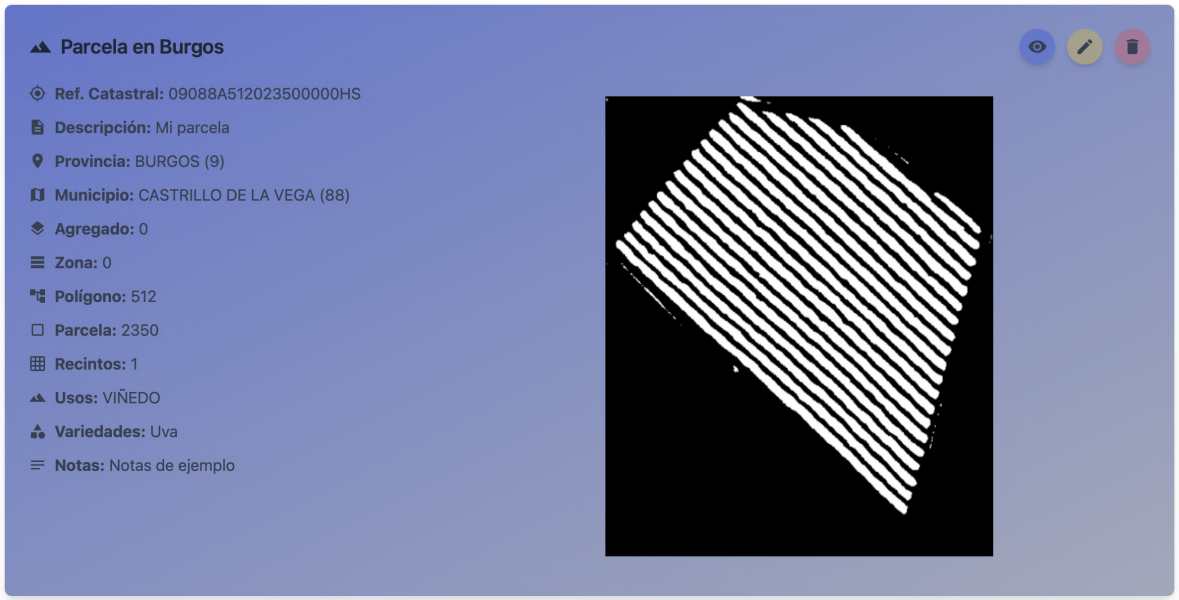

Figura 8.25: Manual de usuario — Viñedos (2) (Detalle de parcela)

Si se pulsa de nuevo, se mostrarán cajas señalando las hileras de viñedos (Figura 8.26).

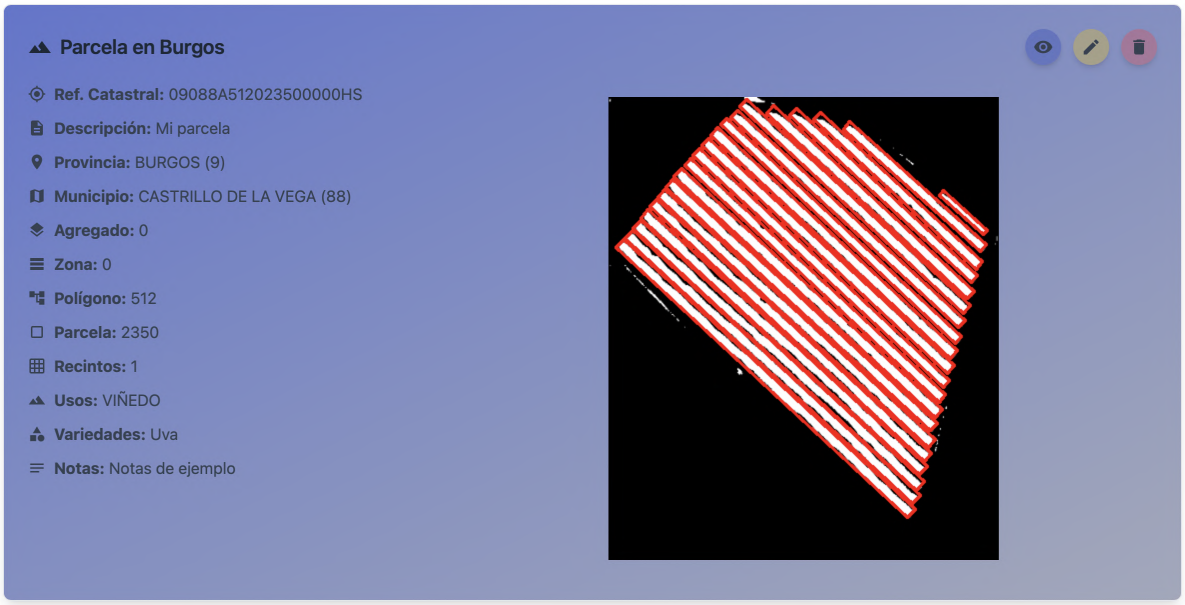

Figura 8.26: Manual de usuario — Viñedos (3) (Detalle de parcela)

Pulsando de nuevo el botón se volverá a la imagen original: el recorte satelital de la parcela.

En cuanto al *Inspector de recintos*, este botón también estará presente con la misma implementación para cada uno de los recintos de la parcela. Además, se muestran los datos obtenidos por el modelo de IA, así como diferentes métricas económicas calculadas a partir de ellos (Figura 8.27).

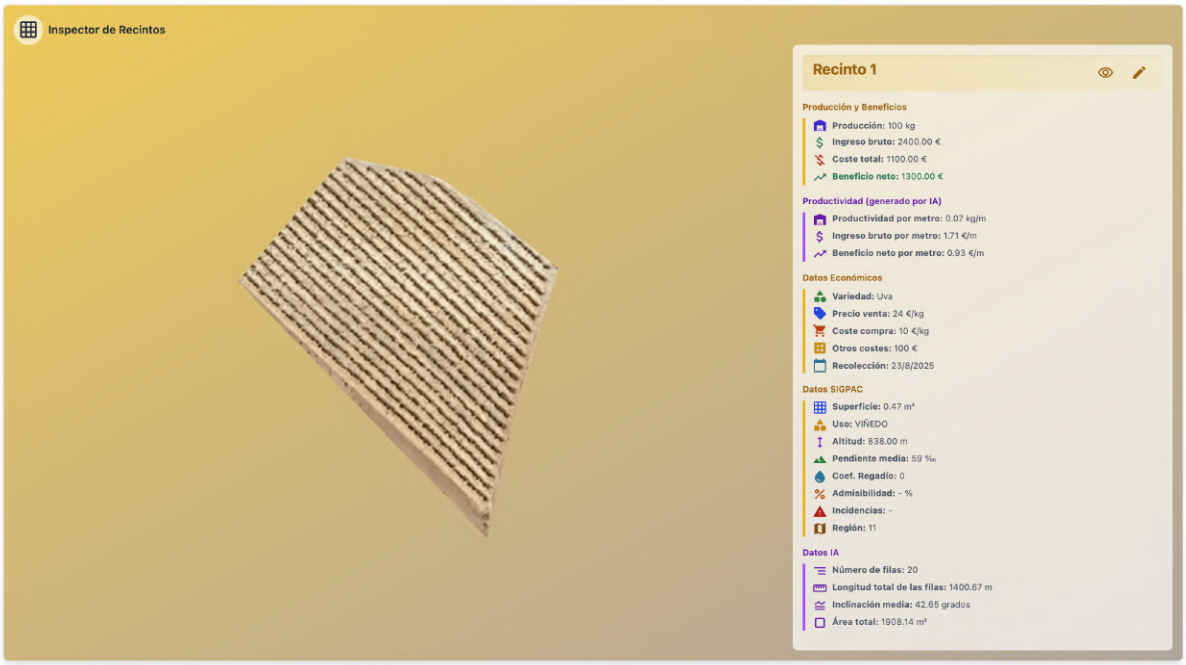

Figura 8.27: Manual de usuario — Viñedos (4) (Detalle de parcela)

8.6. Visor de parcelas

El **Visor de parcelas** permite al usuario explorar gráficamente todas las parcelas registradas en España. La interfaz del visor (ver Figuras 8.28 y 8.29) se compone de los siguientes bloques:

- **Mapa interactivo:** se muestra un mapa satelital implementado con la librería *Leaflet*. Sobre él se dibujan recintos mediante capas vectoriales (polígonos), diferenciados con colores y contornos.
- **Panel de la derecha:** para buscar parcelas mediante referencia catastral. Además, al seleccionar una parcela, en este panel se muestra su información y diferentes opciones.
- **Panel de la izquierda:** diferentes opciones relacionadas con el mapa.
  - Modo de selección: selecciona parcelas enteras (todos sus recintos), o recintos individualmente. Opción por defecto: `Parcelas`.
  - Máx. de parcelas mostradas: cuanto mayor sea, peor rendimiento. Opción por defecto: `200`.
  - Mostrar mis parcelas: muestra en el mapa las parcelas del usuario coloreadas en verde. Opción por defecto: `desactivado`.

  Además, mediante la pestaña de *MIS PARCELAS* el usuario puede ver en el mapa rápidamente cualquiera de sus parcelas guardadas.

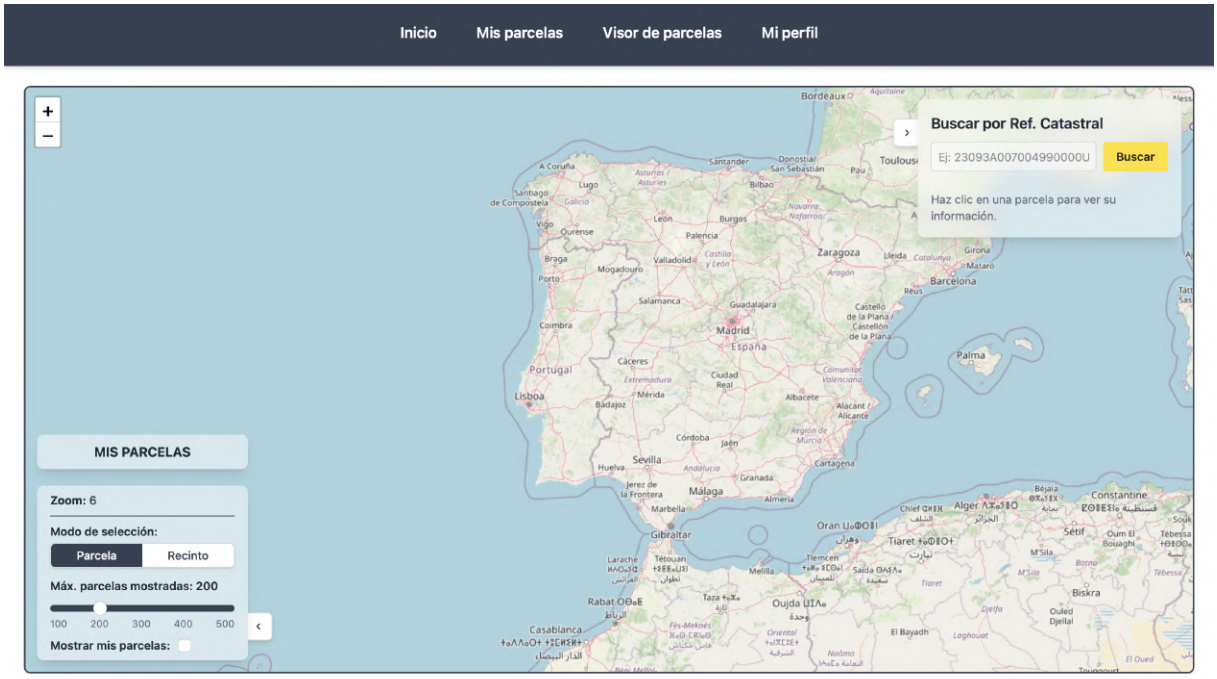

Figura 8.28: Manual de usuario — Visor de parcelas (1)

Recorrido guiado por el visor de parcelas

Pasando el ratón por encima de la pestaña *MIS PARCELAS* ubicada en el panel de la izquierda, se muestran las parcelas del usuario (Figura 8.29).

Figura 8.29: Manual de usuario — Visor de parcelas (2)

Pulsando en una de ellas, se selecciona la parcela y el mapa se centra en ella, apareciendo coloreada de azul. Nótese que los recintos ahora visibles aparecen delimitadas por líneas azules (Figura 8.30).

En el panel de la derecha aparece información de la parcela y dos botones: el icono de "objetivo" para centrar la vista del mapa en la parcela; y el icono de cruz para deseleccionar la parcela.

Además, en este caso, como es una parcela guardada por el usuario, también se muestra en verde el mensaje "Parcela guardada", y un botón a la derecha de este para acceder rápidamente al detalle de la parcela.

Figura 8.30: Manual de usuario — Visor de parcelas (3)

Deseleccionando la parcela anterior (botón de cruz del panel derecho) y activando la opción *Mostrar mis parcelas* (*checkbox* del panel izquierdo), en el mapa se muestra la parcela anterior coloreada de verde (Figura 8.31).

Figura 8.31: Manual de usuario — Visor de parcelas (4)

Se desactiva de nuevo esta opción (como por defecto) y se cambia al modo de selección `Recinto` (panel izquierdo). Los recintos en el mapa ahora aparecen delimitados por líneas amarillas en vez de azules. Si se selecciona un recinto cualquiera, este aparece coloreado de amarillo; y su información, tanto la de la parcela a la que pertenece como la específica

del recinto, aparece en el panel de la derecha. Debajo de la información se ubica un boton "Seleccionar toda la parcela" (Figura 8.32).

Figura 8.32: Manual de usuario — Visor de parcelas (5)

Pulsando el botón "Seleccionar toda la parcela" se cambia automáticamente el modo de selección de `Recinto` a `Parcela` (opción por defecto), y la selección pasa del recinto específico a la parcela que contenía el recinto (Figura 8.33). Ahora en el panel de la derecha solo aparece información de la parcela, junto a un botón "Asignar a mi cuenta".

Figura 8.33: Manual de usuario — Visor de parcelas (6)

Pasando el ratón por los diferentes recintos que conforman la parcela se muestran

tarjetas con la información específica de cada recinto (Figura 8.34).

Figura 8.34: Manual de usuario — Visor de parcelas (7)

Deseleccionando la parcela y haciendo *zoom out*, aparece un mensaje en pantalla indicando que hay demasiadas parcelas en el área seleccionada (Figura 8.35).

Figura 8.35: Manual de usuario — Visor de parcelas (8)

Aumentando el número máximo de parcelas mostradas mediante el selector del panel de la izquierda, el mensaje se oculta y de nuevo aparecen recintos en el mapa (Figura 8.36).

Figura 8.36: Manual de usuario — Visor de parcelas (9)

Se reduce el número máximo de parcelas mostradas a su valor por defecto (200) y se intenta buscar una parcela por su referencia catastral. En este caso, se fuerza un error (Figura 8.37).

Figura 8.37: Manual de usuario — Visor de parcelas (10)

Ahora se selecciona una parcela aleatoria y se pulsa el botón “Asignar a mi cuenta”. En ese momento el backend genera el recorte imagen satelital de la parcela y sus recintos. Mientras tanto, se muestra una pantalla de carga (Figura 8.38).

Figura 8.38: Manual de usuario — Visor de parcelas (11)

Cuando se termina de generar el recorte, este se muestra junto a un formulario que el usuario debe rellenar con los datos personalizados de la parcela: nombre, descripción y notas (Figura 8.39).

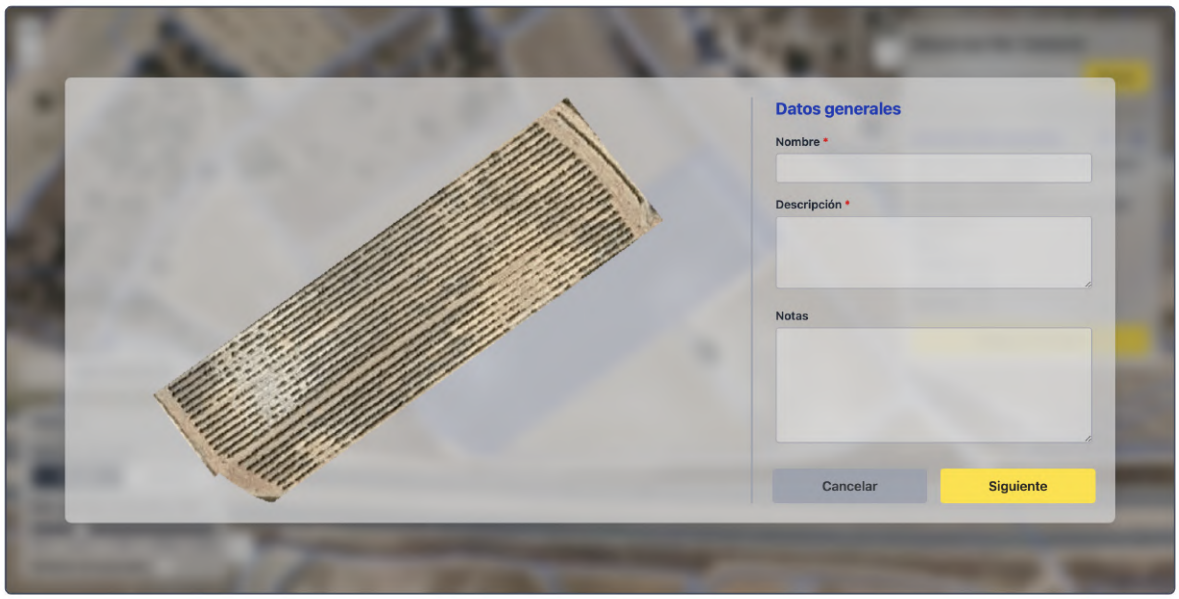

Figura 8.39: Manual de usuario — Visor de parcelas (12)

Si el usuario introduce contenido malicioso o no rellena algún campo obligatorio, se le notifica del error y no se le deja avanzar al siguiente paso (Figura 8.40).

Figura 8.40: Manual de usuario — Visor de parcelas (13)

El usuario ahora deberá completar los datos económicos de cada recinto. A la izquierda del formulario aparece un recorte de la imagen satelital del recinto en cuestión (Figura 8.40).

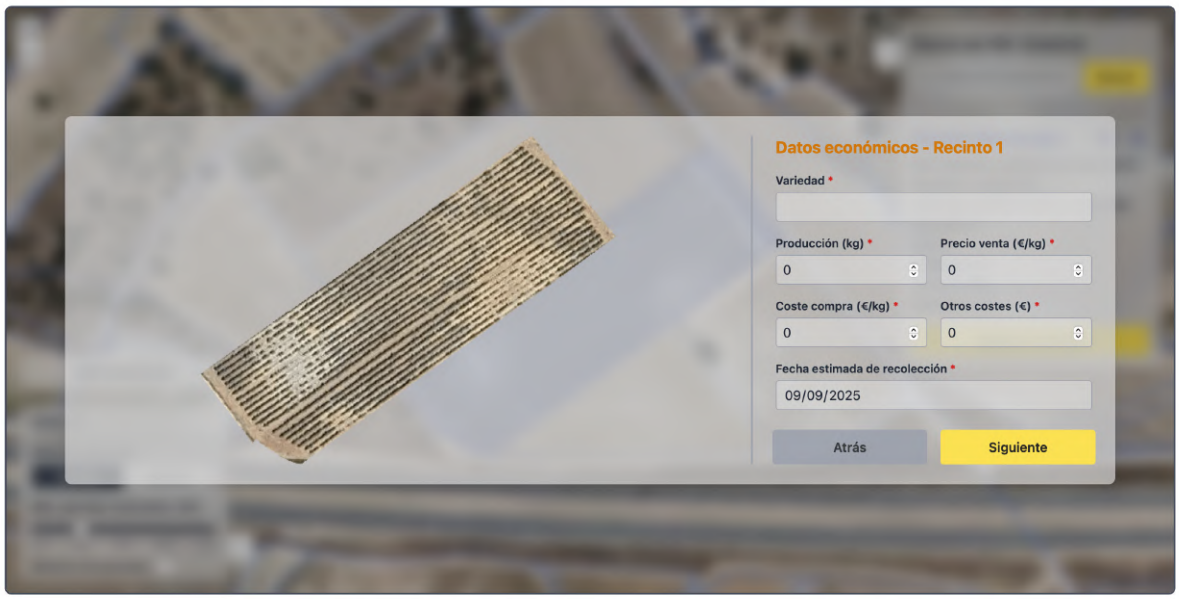

Figura 8.41: Manual de usuario — Visor de parcelas (14)

Esta parcela en concreto consta de dos recintos, por lo que el usuario debe rellenar los datos económicos del segundo recinto. Nótese como la imagen del recinto ha cambiado (Figura 8.41).

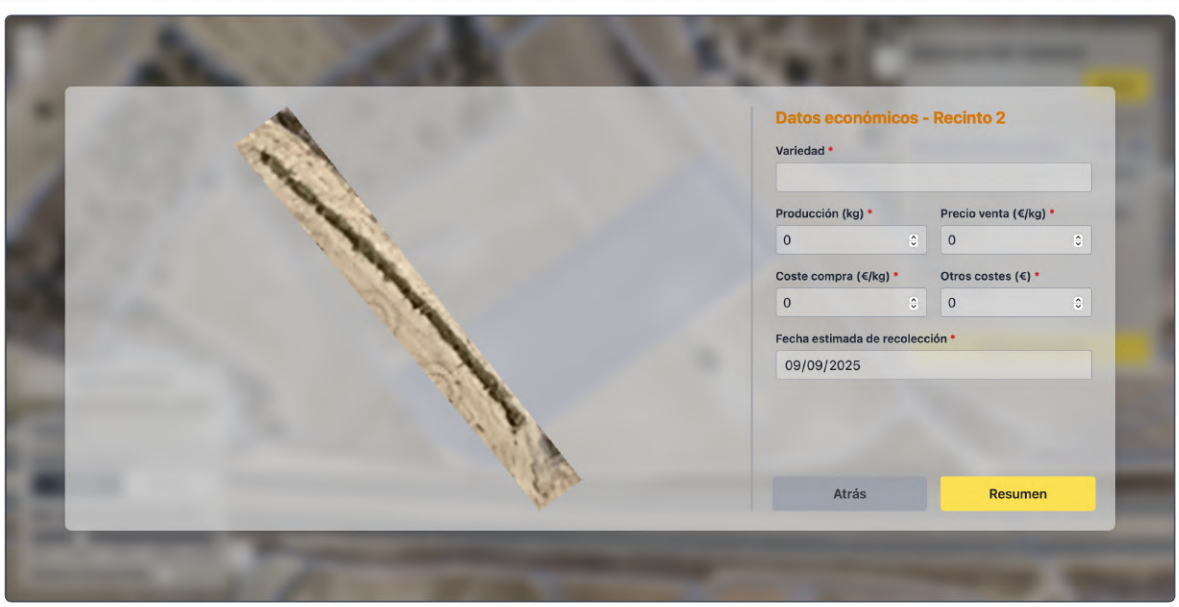

Figura 8.42: Manual de usuario — Visor de parcelas (15)

Tras completar todos los datos necesarios, se muestra una última pantalla a modo resumen. El usuario puede decidir si guardar finalmente la parcela, o volver a las pantallas anteriores para modificar algún dato introducido (Figura 8.42).

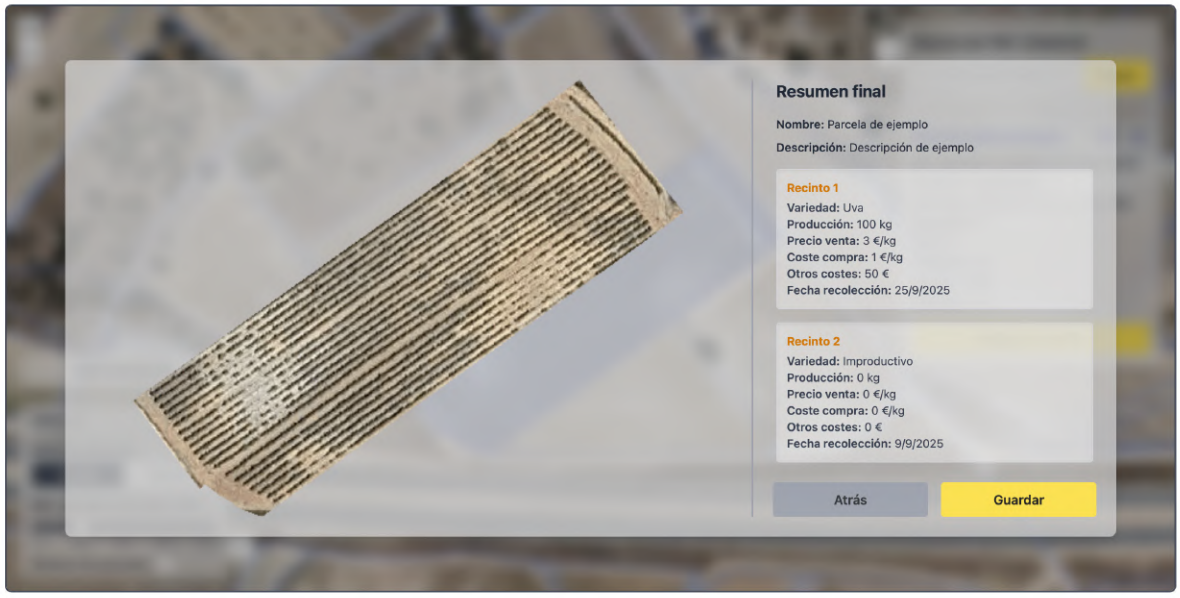

Figura 8.43: Manual de usuario — Visor de parcelas (16)

Si finalmente el usuario decide guardar la parcela, se muestra un mensaje de éxito que desaparece tras unos segundos.

Figura 8.44: Manual de usuario — Visor de parcelas (17)

Nótese cómo durante todas implementaciones mostradas en esta pantalla (Visor de parcelas), se relaciona el color amarillo con recintos y azul con parcelas. Esto es coherente con las demás pantallas explicadas anteriormente, donde los paneles que se refieren a parcelas son azules y los referidos a recintos amarillos.

8.7. Mi perfil

En la sección **Mi perfil** (ver Figura 8.45) el usuario puede consultar y modificar la información básica asociada a su cuenta.

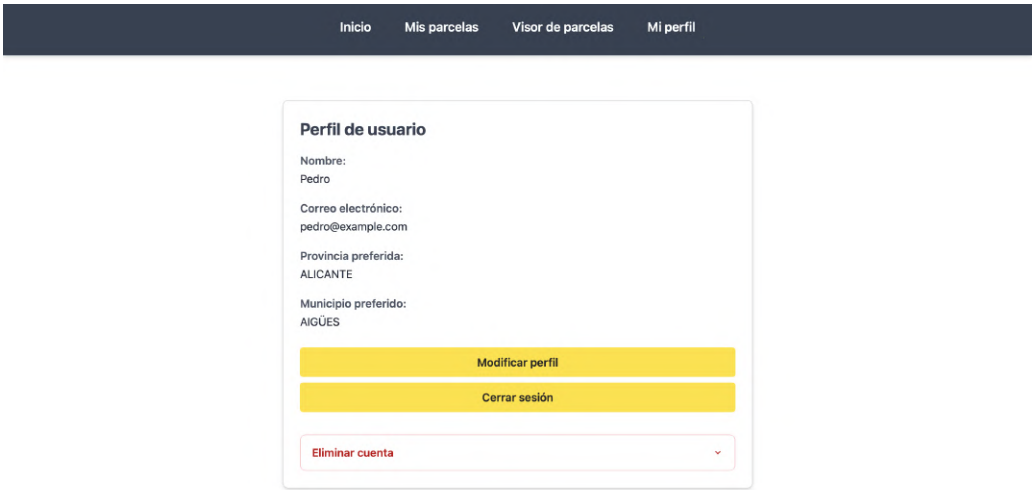

Figura 8.45: Manual de usuario — Mi perfil (1)

Si se pulsa en el botón “Modificar perfil”, el usuario puede modificar todos sus datos (Figura 8.46).

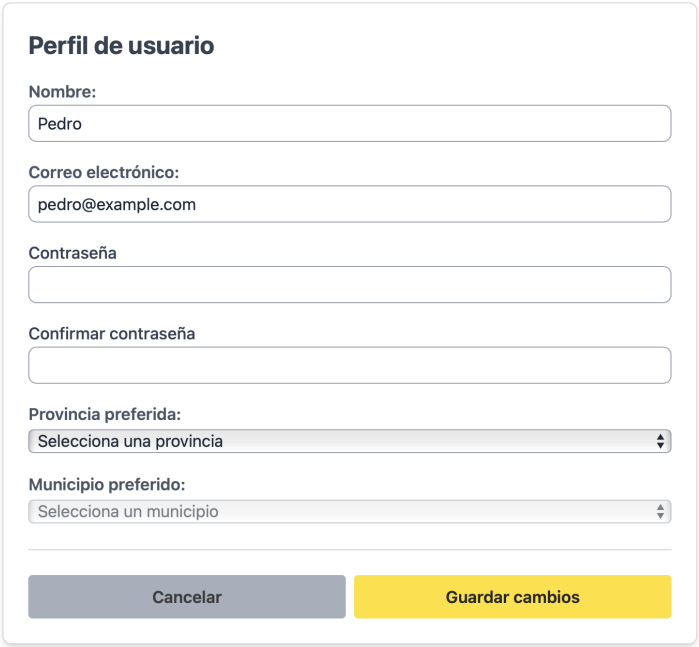

Figura 8.46: Manual de usuario — Mi perfil (2)

El usuario puede cerrar la sesión pulsando en el botón “Cerrar sesión”. Al pulsar, se muestra un modal de confirmación (Figura 8.47).

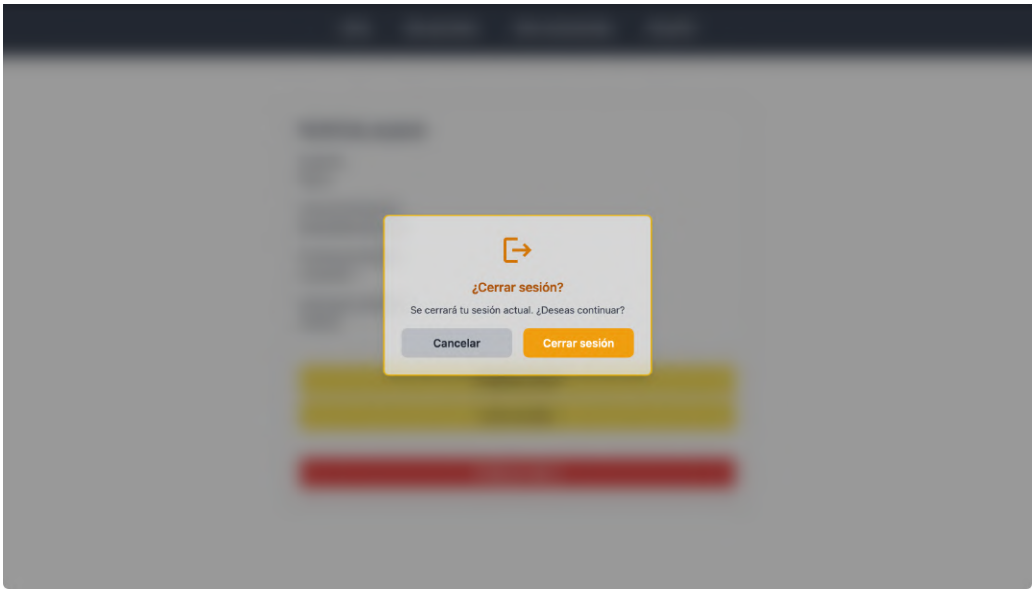

Figura 8.47: Manual de usuario — Mi perfil (3)

El usuario puede eliminar por completo su cuenta pulsando en el *dropdown* “Eliminar cuenta”. Al pulsar, se muestra un mensaje de advertencia con un botón de confirmación (Figura 8.48). Si se pulsa este último botón, por se muestra de nuevo un modal de confirmación (Figura 8.49).

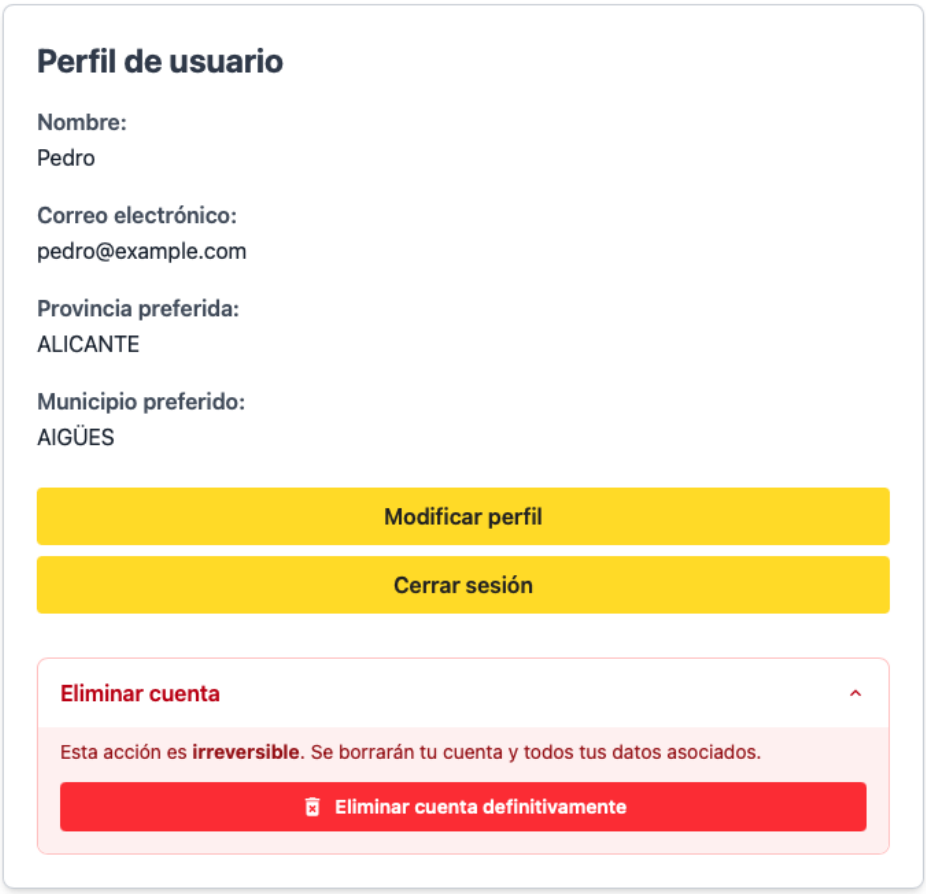

Figura 8.48: Manual de usuario — Mi perfil (4)

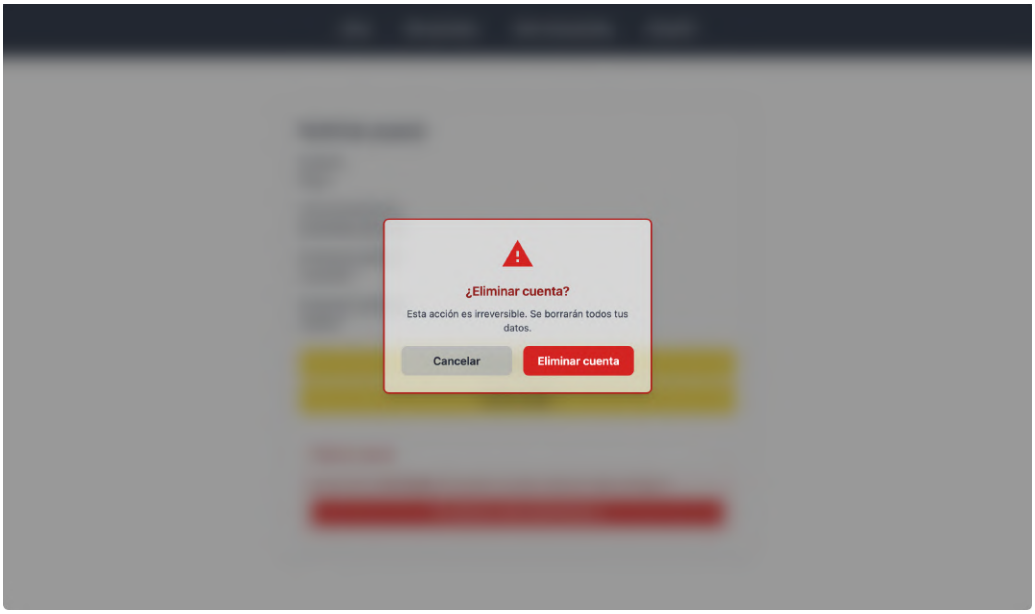

Figura 8.49: Manual de usuario — Mi perfil (5)

Capítulo 9

Conclusiones y trabajo futuro

El desarrollo de esta aplicación ha supuesto un arduo, aunque gratificante, desafío personal. En este tipo de proyectos, el desarrollador tiende a mirar constantemente hacia un futuro, un final, que no llega. La implementación de una función vistosa y útil para el usuario requiere, primero, de una buena estructuración y manejo interno de los datos básicos que servirán como materia primera, no solo de esa implementación en especifico que el usuario tanto desea (y el desarrollador tanto ansía ver en funcionamiento), sino de todas las demás.

En numerosas ocasiones uno tiene la percepción de que, por más horas de su tiempo que emplee en el desarrollo, siempre surgen nuevos problemas y obstáculos que eternizan el proceso y emborronan la linea de meta, situándola cada vez más lejos. Es precisamente en estos momentos en los que se debe mantener una actitud de perfección y resiliencia ya que, una vez superados estos obstáculos, la base construida, tanto referida a la aplicación como a uno mismo, es lo suficientemente buena como para que todo lo que venga inmediatamente sea de resolución simple y directa.

En palabras simples y populares: "No empieces la casa por el tejado".

9.1. Dificultades y aprendizaje

Más allá de las obvias dificultades y aprendizajes técnicos que supone el desarrollo full-stack de una aplicación de estas características, véase el manejo de JavaScript (en este caso TypeScript), React, NestJS, PostgreSQL..., sin duda la principal dificultad a la que me he enfrentado en este proceso ha sido la mencionada en los párrafos anteriores: la gestión emocional y temporal, con su (deseable) consiguiente aprendizaje.

Si tuviera que destacar otro par dificultad-aprendizaje, sin pensarlo dos veces me decantaría por la búsqueda y entendimiento de la información especifica del campo en el que se desarrolla la aplicación, en este caso la agricultura. A su vez, de nuevo la búsqueda y entendimiento del "universo imaginario" en el que los usuarios objetivo interactúan con la aplicación, con objeto de vislumbrar cuáles son las implementaciones que generan más valor.

Para esto último, en una gran empresa se pasa de la imaginación a la realidad, llevando a cabo pruebas y cuestionarios reales entre los usuarios objetivo. En el caso de este proyecto estas (recomendables) prácticas, como cabe esperar, se escapan en términos de tiempo y alcance.

9.2. Posibles mejores y líneas de evolución

El proyecto llevado a cabo sirve de base y esqueleto para numerosas futuras implementaciones con las que se alcanzarían nuevas cotas de funcionalidad y experiencia de usuario. Algunas de estas posibles implementaciones y líneas de evolución serían:

- **Apartado de Análisis:** Añadir un apartado específicamente centrado en el análisis económico permitiría un mayor desglose de estos datos y supondría mejor toma de decisiones para el usuario. En este apartado, se podrían implementar diversas métricas (por ejemplo, el porcentaje de ingresos que representa cada parcela) y gráficas con el objetivo de aumentar el valor añadido de la aplicación.
- **Mejora de los datos económicos:** Si bien actualmente válidos, una mayor granularidad según el tipo de cultivo de cada recinto enriquecería y haría más especifica la aplicación.
- **Mejora de recorte de imágenes satelitales:** El script de recorte de imágenes satelitales a veces falla en parcelas grandes o con recintos separados.
- **Sistema de alertas:** Implementar un sistema de alertas configurable que avise al usuario cuando su producción, ingresos o cualquier otra métrica personalizable cae por debajo de cierto umbral (o se encuentra por encima de cierto límite).
- **Inteligencia Artificial para todos los cultivos:** Actualmente, el modelo de IA implementado solo funciona con (y está diseñado para) viñedos; y los datos ofrecidos podrían ser más útiles. Un modelo que funcione con todo tipo de cultivos y que prediga la producción y otros datos económicos basados, no solo en la imagen satelital de la parcela, sino en las condiciones meteorológicas o el precio de mercado de los insumos aumentaría extraordinariamente el valor añadido de la aplicación.
- **Personalización de los contornos de las parcelas (y recintos):** Aunque actualmente se emplean datos de fuentes oficiales, en la realidad estos límites son más difusos y usualmente acordados extraoficialmente por los agricultores.
- **Aplicación móvil:** Si bien la aplicación está diseñada para adaptarse a todo tipo de pantallas y dispositivos, es decir, es responsive, una app específica para dispositivos iOS y Android sería de gran utilidad para los usuarios.
- **Funcionalidades de red social:** Aumentar la personalización de los perfiles de usuario e implementar chats y búsqueda de perfiles permitiría a los usuarios comunicarse entre sí y posiblemente llevar a cabo transacciones entre ellos, dentro o fuera de la aplicación.

Estas posibles mejoras y líneas de evolución muestran el enorme potencial de crecimiento de la plataforma. La incorporación de análisis económicos avanzados, sistemas de alerta, IA para todos los tipos de cultivo, flexibilidad en la definición de contornos y un salto a la experiencia móvil o social convertirían este prototipo en una solución de referencia para la agricultura de precisión. Cada una de estas mejoras refuerza la misión de ofrecer a técnicos y gestores agrarios una herramienta integral, ágil y orientada al dato, capaz de adaptarse a los retos de un sector cada vez más digital y conectado.

Capítulo A

Endpoints externos

Este anexo recoge los principales *endpoints* consumidos por el frontend de la aplicación, organizados por proveedor. Para cada recurso se indica su propósito, parámetros relevantes y el formato de respuesta esperado.

SIGPAC — Códigos de referencia

`https://sigpac-hubcloud.es/codigossigpac`

- `.../provincia.json`
  Devuelve el listado de provincias. Respuesta: JSON.
- `.../municipio{provincia}.json`
  Devuelve los municipios de una provincia dada. Respuesta: JSON.
- `.../cod_uso_sigpac.json`
  Catálogo de usos SIGPAC. Respuesta: JSON.
- `.../cod_aprovechamiento.json`
  Catálogo de aprovechamientos. Respuesta: JSON.
- `.../cod_incidencia.json`
  Catálogo de incidencias. Respuesta: JSON.

Para más información consultar [10].

SIGPAC — OGC API (GeoJSON)

`https://sigpac-hubcloud.es/ogcapi/collections/recintos`

- `.../items?bbox={bbox}&limit={limit}`
  Recintos en el área delimitado por `bbox`. Devuelve como máximo `limit` recintos. Respuesta: `GeoJSON`.
- `.../items/{id}`
  Consulta un recinto específico dado su ID. Respuesta: `GeoJSON`.

Para más información consultar [12].

SIGPAC — Servicios de consulta

`https://sigpac-hubcloud.es/servicioconsultassigpac/query`

- `.../refcatparcela/{pr}/{mu}/{ag}/{zo}/{po}/{pa}.json`
  Obtiene la referencia catastral a partir de los códigos SIGPAC (provincia, municipio, agregado, zona, polígono, parcela). Respuesta: JSON.
- `.../recinfoparc/{pr}/{mu}/{ag}/{zo}/{po}/{pa}.geojson`
  Devuelve en `GeoJSON` los recintos de una parcela concreta.
- `.../recinfo/{pr}/{mu}/{ag}/{zo}/{po}/{pa}/{re}.{formato}`
  Detalle de un recinto `re` específico, en JSON o GeoJSON.
- `.../recinfobypoint/{srid}/{x}/{y}.{formato}`
  Consulta inversa: devuelve el recinto en el que cae un punto (coordenadas).

Para más información consultar [11].

Catastro — Servicios OVC

`https://ovc.catastro.meh.es/OVCServWeb/OVCWcfCallejero`

- `.../COVCCallejero.svc/json/Consulta_DNPRC?RefCat={refcat}`
  Devuelve la información asociada a una referencia catastral. Respuesta: JSON.
- `.../COVCCoordenadas.svc/json/Consulta_CPMRC?RefCat={refcat}`
  Devuelve las coordenadas geográficas asociadas a una referencia catastral. Respuesta: JSON.

Para más información consultar [1].

Open-Meteo — Predicción meteorológica

`https://api.open-meteo.com/v1`

- `.../forecast?latitude={lat}&longitude={lon}&daily=...`
  Predicción diaria. Se añade por parámetro la información que se quiera obtener. Formato: JSON.
- `.../forecast?latitude={lat}&longitude={lon}&hourly=...`
  Predicción horaria. Se añade por parámetro la información que se quiera obtener. Formato: JSON.

Para más información consultar [6].

Bibliografía

[1] Dirección General del Catastro, *Servicios Web – Documentación*, [En línea]. Disponible en: `https://www.catastro.hacienda.gob.es/ws/Webservices_Libres.pdf`.

[2] JSON Web Tokens, *JWT – Introduction*, [En línea]. Disponible en: `https://jwt.io/introduction/`.

[3] Leaflet, *Leaflet – API Reference*, [En línea]. Disponible en: `https://leafletjs.com/reference.html`.

[4] MSW, *Mock Service Worker – Documentación oficial*, [En línea]. Disponible en: `https://mswjs.io/docs/`.

[5] NestJS, *NestJS – Framework para aplicaciones Node.js eficientes y escalables*, [En línea]. Disponible en: `https://docs.nestjs.com/`.

[6] Open-Meteo, *Weather API Documentation*, [En línea]. Disponible en: `https://open-meteo.com/en/docs`.

[7] PostgreSQL Global Development Group, *PostgreSQL Documentation*, [En línea]. Disponible en: `https://www.postgresql.org/docs/`.

[8] React, *React – Documentación oficial*, [En línea]. Disponible en: `https://es.react.dev/`.

[9] Redux, *Redux – Getting Started*, [En línea]. Disponible en: `https://redux.js.org/introduction/getting-started`.

[10] SIGPAC Hubcloud, *Servicio de códigos SIGPAC*, [En línea]. Disponible en: `https://sigpac-hubcloud.es/html/listCod/descServicio.html`.

[11] SIGPAC Hubcloud, *Servicio CSP de consulta SIGPAC*, [En línea]. Disponible en: `https://sigpac-hubcloud.es/html/csp/descServicio.html`.

[12] SIGPAC Hubcloud, *Servicio OGC API de consulta SIGPAC*, [En línea]. Disponible en: `https://sigpac-hubcloud.es/html/ogc-api/descServicio.html`.

[13] Tailwind Labs, *Tailwind CSS – Documentation*, [En línea]. Disponible en: `https://tailwindcss.com/docs`.

[14] TypeORM, *TypeORM – Getting Started*, [En línea]. Disponible en: `https://typeorm.io/docs/getting-started`.

[15] Microsoft, *TypeScript – Documentación oficial*, [En línea]. Disponible en: `https://www.typescriptlang.org/docs/`.

[16] Vite, *Vite – Build tool*, [En línea]. Disponible en: `https://vitejs.dev/`.

[17] Vittory, J., *JWT Refresh Tokens: cómo implementarlos en tu API*, [En línea]. Disponible en: `https://dev.to/jeanvittory/jwt-refresh-tokens-2g3d`.