



Universidad de Jaén

Escuela Politécnica Superior de Jaén

Aplicación Web para el Análisis Temporal de la Salud del Olivar

Autora: Alba Gómez Liébana

Grado: Ingeniería Informática

Directora: Dña. Lidia Ortega Alvarado
Departamento: Informática

Fecha: 17/06/2024

Licencia CC



CREA



UNIVERSIDAD DE JAÉN

Dña. Lidia Ortega Alvarado, tutora del Trabajo Fin de Grado titulado: **Aplicación Web para el Análisis Temporal de la Salud del Olivar**, que presenta Alba Gómez Liébana, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, junio de 2024

El estudiante

Firmado por GOMEZ LIEBANA
ALBA - ***4675** el día
17/06/2024 con un
certificado emitido por AC
FNMT Usuarios

La tutora

ORTEGA
ALVARADO
LIDIA -
24241033E

Firmado digitalmente
por ORTEGA
ALVARADO LIDIA -
24241033E
Fecha: 2024.06.17
19:38:38 +02'00'

Alba Gómez Liébana

Dña. Lidia Ortega Al-
varado

Agradecimientos

En primer lugar, quiero expresar mi gratitud a mi tutora Lidia, por ser la guía de este proyecto, por su paciencia y dedicación. Su conocimiento y experiencia que han sido cruciales para la realización de este trabajo. Gracias tanto a ella como a Ruth por estar siempre disponibles para ayudarme y proporcionarme las herramientas necesarias.

A mis padres, quienes siempre han creído en mí más de lo que yo misma lo hacía, y a mis hermanos, gracias por convertir nuestro hogar en una fuente inagotable de amor incondicional.

A mis abuelos, por su amor ilimitado y sabiduría inestimable. En especial a mi abuela Julia, que siempre ha estado tan orgullosa de mí.

A mi pareja, por su cariño, comprensión y paciencia, y representar en todas sus variantes y significados la palabra "amor".

Y, por último, a la Alba de hace 4 años, que no se imaginaría nunca hasta donde ha llegado ni la mujer en la que se ha convertido.

Gracias a todos.

Tabla de contenidos

1. Introducción	1
1.1. Motivación	2
1.2. Contexto del proyecto	3
1.3. Objetivo General	3
1.4. Objetivos Específicos	4
1.5. Estructura de la memoria	4
1.6. Normas y referencias	5
1.6.1. Mecanismos de control de calidad	6
1.7. Antecedentes y estado del arte	6
1.7.1. Antecedentes	6
1.7.2. Estado del Arte	8
1.7.3. Imágenes Satelitales	17
1.7.4. Enfermedades del olivar	20
2. Planificación	29
2.1. Metodología de desarrollo	29
2.1.1. Scrum	30
2.1.2. Método iterativo/incremental	32

2.2. Identificación de tareas	32
2.2.1. Primera Iteración: Estudio y gestión de datos de entrada	32
2.2.2. Segunda Iteración: Diseño de la base de datos	33
2.2.3. Tercera Iteración : Arquitectura del sistema y pruebas	33
2.2.4. Cuarta Iteración: Interfaz de Usuario	34
2.2.5. Quinta Iteración: Análisis de los datos	35
2.3. Planificación temporal	35
2.3.1. Diagrama de Gantt	36
2.4. Estimación de costes y viabilidad	37
2.4.1. Estimación de tamaño y esfuerzo	37
2.4.2. Presupuesto	42
3. Análisis del sistema	47
3.1. Análisis de Requisitos	47
3.1.1. Requisitos funcionales	48
3.1.2. Requisitos no funcionales	50
3.1.3. Perfil de usuario	51
3.2. Análisis y diseño del sistema	51
3.2.1. Modelo del dominio	52
3.2.2. Casos de Uso	52
3.2.3. Diagrama de casos de uso	54
3.2.4. Diagramas de secuencia del sistema	56
3.3. Análisis de Riesgos	61
4. Desarrollo del Sistema	63

4.1. Introducción	63
4.1.1. Descripción de la solución propuesta	64
4.2. Trabajo inicial	65
4.3. Primera iteración: Estudio y gestión de datos de entrada	65
4.3.1. Introducción	65
4.3.2. Obtención de Fincas y Olivos	70
4.3.3. Obtención de Factores	77
4.3.4. Conversión de datos	85
4.4. Segunda iteración: Diseño de la Base de Datos	90
4.4.1. Introducción	90
4.4.2. Diseño de la Base de Datos	93
4.5. Tercera iteración: Arquitectura del sistema y pruebas	106
4.5.1. Introducción	106
4.5.2. Capa de Lógica de Negocio	113
4.5.3. Capa de Persistencia	115
4.5.4. Capa de Servicios	124
4.6. Cuarta iteración: Diseño de la Página Web	128
4.6.1. Introducción	128
4.6.2. Diseño de la Interfaz de Usuario	131
4.6.3. Implementación de la Interfaz de Usuario	139
4.7. Quinta iteración: Análisis de los Datos	159
4.7.1. Análisis Datos Finca	159
4.7.2. Análisis Datos Olivo	162
4.8. Pruebas finales con usuarios	163

4.8.1. Resultados perfil informático	164
4.8.2. Resultados perfil técnico agrícola	164
5. Conclusiones y trabajos futuros	167
5.1. Estudio de comercialización	167
5.1.1. Objetivos	167
5.1.2. Análisis del Entorno	168
5.1.3. Plan de Mercadotecnia	168
5.2. Conclusiones	169
5.3. Trabajo futuro	170
Bibliografía	v
6. Apéndices	VI
6.1. Guía original del Trabajo Fin de Título	VI
6.2. Manuales de usuario	X
6.2.1. Página de inicio	XI
6.2.2. Página Enfermedades	XI
6.2.3. Página Índices de Vegetación	XII
6.2.4. Página Finca Específica	XII
6.2.5. Página Olivo Específico	XIII

Lista de figuras

1.1. Mar de olivos en Jaén - Aceitunas Guerra	2
1.2. Imagen ilustrativa de CropX - CropX	7
1.3. Logo de Treemetrics - Treemetrics	7
1.4. Imagen ilustrativa de SmartOlive - SmartPartner	8
1.5. Interfaz de la plataforma SIGPAC del Gobierno de España,	9
1.6. Espectro Electromagnético - Concepto	11
1.7. Ejemplo de índices de vegetación - Geoinnova	12
1.8. Ejemplo NDVI Pseudocolor Monobanda - Elaboración propia	12
1.9. Ejemplo SAVI Pseudocolor Monobanda - Elaboración propia	14
1.10. Ejemplo NDMI Invierno 2022 - Elaboración propia	15
1.11. Sentinel-2 - European Space Agency	18
1.12. MODIS - NASA Ocean Color	19
1.13. Vista ERA5-Land	20
1.14. Euzophera Pinguis - Process Olive	20
1.15. Bactrocera oleae Gmel - Junta de Andalucía	22
1.16. Spilotea Oleagina - Probelte	23
1.17. Prays oleae Bern - Agrotropiche	24
1.18. Verticillium dahliae Kleb - ABC Sevilla	26

1.19. Tabla de riesgos de enfermedades - Elaboración Propia	28
2.1. Modelo CVC - Elaboración propia	30
2.2. Scrum Framework	31
2.3. Diagrama de Gantt - Elaboración Propia	36
2.4. Parámetros COCOMO - Elaboración propia	38
2.5. Valores constantes COCOMO - Elaboración propia	39
2.6. Factores COCOMO - Elaboración propia	40
2.7. Amortización del Hardware - Elaboración Propia	43
2.8. Amortizacion Software - Elaboración Propia	44
2.9. Coste Personal - Elaboración Propia	44
3.1. Programa Visual Paradigm	52
3.2. Modelo del Dominio - Elaboración Propia	53
3.3. Elementos de un Caso de Uso - Elaboración propia	55
3.4. Diagrama Casos de Uso - Técnico Agrícola	56
3.5. Elementos de un Diagrama de Secuencia - Elaboración Propia	58
3.6. Diagrama de Secuencia Buscar Finca - Elaboración Propia	59
3.8. Diagrama de Secuencia Ver Datos Enfermedades - Elaboración Propia	59
3.7. Diagrama de Secuencia Buscar Olivo - Elaboración Propia	60
3.9. Diagrama de Secuencia Ver Información Índices de Vegetación - Elaboración Propia	60
3.10. Tabla Análisis de Riesgos - Elaboración Propia	62
4.1. Tecnologías usadas - Elaboración propia	63
4.2. Diagrama UML Transformador - Elaboración propia	66

4.3. IntelliJ Logo	68
4.4. Google Earth Engine - Gis Beers	68
4.5. QGis	69
4.6. Pentaho Spoon	69
4.7. Imagen de alto contraste J2 - Grupo de Investigación	71
4.8. GEU Ejemplo - Elaboración Propia	72
4.9. Umbralización Índice SAVI - Elaboración propia	74
4.10. Ejemplo de contorno de finca en QGis - Elaboración Propia	75
4.11. Ejemplo de recorte de olivos en QGIS - Elaboración Propia	76
4.12. Ejemplo de dilatación de olivos en QGIS - Elaboración Propia	76
4.13. Ejemplo descarga de geometría SigPac - Elaboración Propia	77
4.14. Carga de datos en Google Earth Engine - Elaboración Propia	78
4.15. Obtención datos de dron - Elaboración Propia	84
4.16. Pasos para transformar XLSL en JSON Pentaho Spoon - Elaboración Propia	88
4.17. Resultado Temperatura JSON - Elaboración Propia	88
4.18. Resultado Índices JSON - Elaboración Propia	90
4.19. Base de Datos Inicial - Elaboración propia	91
4.20. Tabla Provincia - Elaboración Propia	94
4.21. Tabla Municipio - Elaboración Propia	94
4.22. Tabla Zona - Elaboración Propia	95
4.23. Tabla Finca - Elaboración Propia	96
4.24. Tabla Cosecha - Elaboración Propia	97
4.25. Tabla Histórico Finca - Elaboración propia	98

4.26. Tabla Objeto - Elaboración propia	99
4.27. Tabla Histórico Datos - Elaboración Propia	100
4.28. Diagrama Entidad Relación - Elaboración Propia	101
4.29. Frameworks Java	107
4.30. Logo Spring Framework	108
4.31. Capas de la Arquitectura de las Aplicaciones Empresariales - Elaboración Propia	111
4.32. Arquitectura de la Aplicación - Elaboración Propia	113
4.33. Crear Conexión Base de Datos - Elaboración Propia	115
4.34. Vue JS - Lenguaje JS	130
4.35. Wireframe Página Principal - Elaboración Propia	132
4.36. Wireframe Página Fincas - Elaboración Propia	133
4.37. Wireframe Página Finca Específica - Elaboración Propia	133
4.38. Wireframe Página Olivo Específico - Elaboración Propia	134
4.39. Wireframe Página Índices Vegetación - Elaboración Propia	134
4.40. Wireframe Página Tabla Enfermedades - Elaboración Propia	135
4.41. Mock-Up Página Principal - Elaboración Propia	136
4.42. Mock-Up Página Fincas - Elaboración Propia	136
4.43. Mock-Up Página Finca Específica - Elaboración Propia	137
4.44. Mock-Up Página Olivo Específico - Elaboración Propia	137
4.45. Mock-Up Página Índices Vegetación - Elaboración Propia	138
4.46. Mock-Up Página Tabla Enfermedades - Elaboración Propia	138
4.47. Paleta de Colores Interfaz de Usuario - Elaboración Propia	139
4.48. Página Home de la Aplicación Web - Elaboración Propia	143

4.49. Página Enfermedades de la Aplicación Web - Elaboración Propia	145
4.50. Página Índices de Vegetación de la Aplicación Web - Elaboración Propia	146
4.51. Página Buscador Fincas de la Aplicación Web - Elaboración Propia . . .	147
4.52. Página Finca Específica de la Aplicación Web - Elaboración Propia . . .	151
4.53. Ejemplo Mapa Finca - Elaboración Propia	153
4.54. Ejemplo Gráficas Finca - Elaboración Propia	157
4.55. Ejemplo Vista Olivo - Elaboración Propia	159
4.56. Informe Finca J1 - Elaboración Propia	160
4.57. Gráfica Cosechas J1 - Elaboración Propia	160
4.58. Gráficos Índices de Vegetación J1 - Elaboración Propia	161
4.59. Gráfico Precipitaciones J1 - Elaboración Propia	162
4.60. Gráfico Temperaturas J1 - Elaboración Propia	162
4.61. Ejemplo Gráficos Índices de Vegetación Olivo - Elaboración Propia . . .	163
6.1. Diagrama de Recorridos de la Página Web - Elaboración Propia	X
6.2. Página Home - Elaboración Propia	XI
6.3. Página Enfermedades - Elaboración Propia	XII
6.4. Página Índices de Vegetación de la Aplicación Web - Elaboración Propia	XIII
6.5. Página Finca Específica - Elaboración Propia	XIV
6.6. Página Olivo Específico - Elaboración Propia	XV

Lista de listados de código

4.1. Definición de colección de Sentinel-2	78
4.2. Obtención de número de imágenes	79
4.3. Cálculo de índices de vegetación	79
4.4. Añadir capas de índices	79
4.5. Exportación en CSV	80
4.6. Cargar datos de precipitaciones y conversión a milímetros	81
4.7. Obtención Precipitaciones por mes	81
4.8. Cargar datos MODIS	82
4.9. Exportación datos temperatura	83
4.10. Ejemplo de inserción en Historico Finca	103
4.11. Ejemplo de inserción en Histórico Datos	104
4.12. Clase HistoricoFinca Capa Dominio	114
4.13. application.properties	116
4.14. Declaración Drive Oracle en pom.xml	116
4.15. Clase Historico Finca Capa Persistencia	116
4.16. ID de la Clase Historico Finca Capa Persistencia	117
4.17. Clase Objeto Capa Persistencia	118
4.18. Repositorio clase Historico Finca	119

4.19. Declaración SistemaFincas	121
4.20. Ejemplo métodos ServicioFinsa	122
4.21. Declaración SistemaFincas	122
4.22. DTOHistoricoFinsa	124
4.23. Controlador REST	125
4.24. Controlador REST - Peticiones	126
4.25. Archivo vite.config.js	139
4.26. Archivo main.js	140
4.27. Archivo router/index.js	140
4.28. Archivo App.vue	142
4.29. Archivo HomeView.vue	143
4.30. Archivo EnfermedadView.vue	145
4.31. Archivo FincaView.vue	147
4.32. Archivo FincaSIGPAC.vue	148
4.33. Archivo FincaMapa.vue	149
4.34. Archivo FincaEspecifica.vue	151
4.35. Función conversión de coordenadas	152
4.36. Definición capas en Leaflet, onMounted()	153
4.37. Función watch() en Mapa PNOA	154
4.38. Script CosechaGrafica	155
4.39. Script Gráficos Finca	157

Capítulo 1

Introducción

En este capítulo se presentará este Trabajo de Fin de Grado. El objetivo del mismo es tener una clara idea del sistema a desarrollar. El capítulo está organizado en los siguientes apartados:

- **MOTIVACIÓN:** en este apartado se explicará la importancia económica y social del sector olivarero en España, especialmente en Jaén.
- **CONTEXTO DEL PROYECTO:** en este apartado se indicará el contexto del proyecto, hablando de una descripción del entorno actual y dónde se encontraría.
- **OBJETIVO GENERAL:** se explicará el objetivo general, a grandes rasgos, del Trabajo de Fin de Grado.
- **OBJETIVOS ESPECÍFICOS:** delimitaremos los objetivos específicos del mismo.
- **ESTRUCTURA DE LA MEMORIA:** en este apartado se explicará todo el proceso de generación del trabajo mediante la especificación de los distintos capítulos del proyecto.
- **NORMAS Y REFERENCIAS:** en este apartado se detallan las normas, referencias y leyes que se han de tener en cuenta tanto en la elaboración del trabajo como en su posterior puesta en práctica.
- **ANTECEDENTES Y ESTADO DEL ARTE:** en este apartado se explicarán los diferentes sistemas ya existentes para monitorizar la salud del olivar y la importancia de desarrollar herramientas más accesibles que permitan a los agricultores obtener información útil sobre la salud de sus olivares de forma sencilla.

1.1. Motivación

La provincia de Jaén, conocida como la cuna de la economía del olivar, desempeña un papel crucial en la producción de aceite de oliva a nivel nacional e internacional. Es por tanto, un referente clave en la industria oleícola. En este contexto, surge la necesidad de preservar y mejorar la salud de los olivares, asegurando la sostenibilidad y la productividad a lo largo del tiempo, ya que la salud del olivar es de vital importancia para la región.

Jaén posee una superficie de más de 550.000 hectáreas de olivos (Fig 1.1, lo que representa una mayor extensión que la totalidad de algunas provincias españolas. La provincia de Jaén alberga un 25 % de la superficie del olivar en España y un 42 % en Andalucía, lo que resulta en la producción del 50 % del total nacional de aceites de oliva y más del 20 % en todo el mundo [1].



Figura 1.1: Mar de olivos en Jaén - Aceitunas Guerra

El desarrollo de una aplicación web para el análisis temporal de la salud del olivar se plantea como una solución a los problemas actuales que enfrenta la agricultura en la provincia de Jaén. Busca proporcionar a los técnicos agrícolas de Jaén y a la industria olivarera una solución eficiente y tecnológica para la gestión integral de la salud de los olivos. Ya que las enfermedades específicas del olivo y otros factores ambientales pueden impactar significativamente en la salud de los cultivos, afectando la calidad y cantidad de la cosecha

El objetivo principal de esta aplicación es aprovechar las últimas tecnologías, como la teledetección y el análisis de datos, para monitorear y evaluar la evolución temporal de los olivares. La aplicación no solo aspira a ser una herramienta tecnológica, sino un aliado estratégico para los técnicos agrícolas, proporcionándoles información valiosa para optimizar sus prácticas y preservar la salud a largo plazo de los olivares.

La aplicación tendrá un impacto positivo en la economía de Jaén al mejorar la

producción y la calidad del aceite de oliva, lo que redundará en un aumento de los ingresos para los agricultores y la creación de empleo.

1.2. Contexto del proyecto

El proyecto surge como respuesta a las limitaciones en los sistemas actuales de monitorización de la salud del olivar. Actualmente, estos sistemas carecen de interactividad y ofrecen una visión limitada y estática de la salud de los olivares. Para administrar la salud de los olivos, los técnicos agrícolas deben lidiar con procesos manuales y rudimentarios. Esto dificulta la toma de decisiones informadas y la implementación de medidas preventivas a largo plazo.

La falta de plataformas específicas para la gestión y análisis de la salud del olivar agrava la falta de herramientas tecnológicas adecuadas en este ámbito. Debido a esto, los olivos se gestionan de manera fragmentada. Además, no se realiza un seguimiento y evaluación constante de su salud.

En este contexto, el presente proyecto propone desarrollar una aplicación web interactiva y dinámica que aborde estas limitaciones. La aplicación ofrecerá a los usuarios, tanto agricultores experimentados como técnicos agrícolas, una plataforma intuitiva y completa para monitorizar y analizar la salud de los olivares.

Se espera que esta herramienta no solo mejore la eficiencia y la productividad en el sector oleícola, sino que también empodere a los técnicos agrícolas involucrados al proporcionarles una visión integral y a largo plazo de la salud de sus cultivos. La interactividad y la capacidad de análisis avanzado serán características clave de esta aplicación, marcándose como una solución innovadora en un campo que ha carecido de avances tecnológicos significativos hasta el momento.

1.3. Objetivo General

El propósito principal de este trabajo es crear, desarrollar e implementar una aplicación web innovadora para analizar temporalmente la salud del olivar en la zona de Jaén. La aplicación tiene como objetivo ofrecer a los técnicos agrícolas una herramienta práctica y fácil de usar que les permita monitorear de manera continua y detallada a sus olivares a lo largo del tiempo.

La aplicación tiene como objetivo abordar los problemas específicos que enfrentan los productores locales en relación a la variabilidad climática, enfermedades específicas del olivo y otros factores ambientales. El objetivo es brindar a los agricultores una solución completa que les permita anticipar y responder proactivamente a amenazas potenciales, maximizando así la salud y la productividad de sus cultivos.

1.4. Objetivos Específicos

A continuación se expondrán todos los objetivos específicos del proyecto:

1. Desarrollar una aplicación web para el análisis temporal de la salud del olivar, que permita a los usuarios visualizar y analizar la información relacionada con el estado de los olivares a lo largo del tiempo.
2. Establecer una conexión efectiva con una base de datos que almacene los datos relevantes para el análisis de la salud del olivar, asegurando la integridad y la disponibilidad de la información.
3. Implementar mapas interactivos de información que permitan a los usuarios interactuar con los datos geoespaciales de los olivares, facilitando la visualización y el análisis de la distribución geográfica de los cultivos.
4. Incorporar la visualización de índices de vegetación, temperaturas y precipitaciones para proporcionar a los usuarios una herramienta efectiva para evaluar la salud de los olivares en función de la vegetación.
5. Desarrollar tablas de enfermedades que permitan a los usuarios acceder a información detallada sobre las enfermedades más comunes que afectan a los olivares, facilitando la identificación y el diagnóstico de problemas de salud en los cultivos.
6. Realizar un análisis temporal de la evolución de las variables objeto de estudio explicadas anteriormente.

1.5. Estructura de la memoria

El objetivo de este apartado es proporcionar una explicación detallada y clara sobre los diferentes apartados que forman parte de este proyecto. Se pretende esclarecer y

presentar de manera comprensible toda la información relevante acerca de cada una de las secciones que han contribuido al desarrollo de la aplicación web.

Este proyecto, por tanto, se encuentra dividido en ocho apartados:

1. El primer capítulo, en el cual nos encontramos actualmente, pretende introducir al lector al trabajo, así como mostrar información teórica sobre el proyecto a tratar.
2. El segundo capítulo, llamado planificación, pretende explicar la metodología de desarrollo que se ha seguido, así como la planificación temporal y la estimación de costes.
3. En el tercer capítulo, llamado análisis del sistema, se realizará todo el proceso de ingeniería del software para analizar el sistema a realizar. Dentro de este podemos encontrar el análisis de requisitos, el diseño del sistema o el análisis de riesgos.
4. En el cuarto capítulo, se realizará el desarrollo del sistema de forma completa. Esta implementación se ha realizado en cinco iteraciones cuya conclusión serán las pruebas con los usuarios.
5. El quinto capítulo, llamado conclusiones y trabajos futuros, albergará un estudio de comercialización y las conclusiones junto con el trabajo futuro posible a realizar.
6. En el sexto capítulo se presentará toda la bibliografía usada en el proyecto.
7. Por último, en el séptimo capítulo, denominado "Apéndices", aparecerán la instalación y configuración del sistema, los manuales de usuario y la Guía Original del Trabajo Fin de Título.

1.6. Normas y referencias

Este proyecto presenta una estructura y contenido adaptados a la norma UNE 157001.

1.6.1. Mecanismos de control de calidad

Para asegurar la calidad en la redacción de este trabajo, este debe ser claro, conciso y comprensible para todos los involucrados e interesados, así como completo, sin presentar omisiones.

El trabajo no requiere un documento de control de edición y revisión de la documentación porque fue creado por una sola persona y aprobado por una tutora. Se han utilizado mecanismos fundamentales para la verificación de integridad y completitud, como un control de versiones a nivel de documento y un control simple de la trazabilidad de especificaciones y requisitos.

1.7. Antecedentes y estado del arte

En esta sección discutiremos los fundamentos teóricos esenciales del Trabajo y las aplicaciones actuales existentes para el análisis temporal de la salud del olivar.

1.7.1. Antecedentes

En este apartado se realizará un análisis sobre el conocimiento previo ya existente y que ha servido, en muchos casos, de base para este trabajo. Esto nos dará una visión más completa del punto de partida del desarrollo.

En el mercado actual, no existe ninguna aplicación web accesible con la que la podamos comparar, que combine todos los objetivos mencionados anteriormente. Por tanto, esta aplicación es muy innovativa, ya que va a permitir un análisis temporal de la salud del olivar, permitiendo a los usuarios visualizar y analizar la evolución de la salud de sus olivares a lo largo del tiempo, identificando tendencias y patrones que pueden ser indicativos de problemas potenciales.

Además, se presentarán mapas interactivos de información sobre las distintas fincas, ya que la interactividad va a ser un factor clave en nuestra aplicación. Permitirá, de igual modo, visualizar gráficas sobre índices de vegetación, temperaturas y precipitaciones. Por tanto, todos los usuarios, incluso aquellos con poca experiencia en tecnología, podrán acceder y comprender fácilmente la información proporcionada por la aplicación.

A continuación, se van a mostrar algunos antecedentes de la aplicación que no tienen el mismo fin, pero cumplen algunos objetivos marcados:

- **Cropx Agronomic Farm Management System (Fig 1.2):** es un sistema de gestión agrícola potente y fácil de usar que utiliza inteligencia artificial y aprendizaje automático para conectar datos agrícolas. Esta aplicación nos ofrece una solución no tan específica para la gestión de cultivos en general. No usa mapas interactivos para la visualización de datos ni tampoco se centra en las enfermedades del olivar.



Figura 1.2: Imagen ilustrativa de Cropx - Cropx

- **TreeMetrics (Fig 1.3):** nuestra aplicación se centra en el análisis de datos históricos y actuales, mientras que esta se enfoca principalmente en la predicción de eventos futuros. En este trabajo se ofrece una plataforma web para la visualización y análisis de datos, mientras que Treemetrics proporciona una API para la integración con otros sistemas. De nuevo, nuestra aplicación es específica para el olivar, mientras que esta puede usarse para otros tipos de cultivo, no siendo tan específica.



Figura 1.3: Logo de Treemetrics - Treemetrics

- **SmartOlive (Fig 1.4):** se trata de un software que optimiza la gestión, elaboración y envasado de la aceituna. Nuestra aplicación se centra en el análisis temporal de la salud del olivar, mientras que SmartOlive ofrece un sistema de apoyo a la decisión basado en inteligencia artificial que se basa principalmente en recomendaciones personalizadas.



Figura 1.4: Imagen ilustrativa de SmartOlive - SmartPartner

1.7.2. Estado del Arte

Sistema de Información Geográfica de Parcelas Agrícolas

El **Sistema de Información Geográfica de Parcelas Agrícolas (SIGPAC)**, permite identificar geográficamente las parcelas declaradas por agricultores y ganaderos en cualquier régimen de ayudas relacionado con la superficie cultivada o aprovechada por el ganado [2].

El SIGPAC, creado originalmente para ayudar a los agricultores a presentar solicitudes con soporte gráfico y realizar controles administrativos y sobre el terreno, se ha convertido en una herramienta muy útil en varios campos agrarios. Esto se debe a su concepción y desarrollo, en el que se utilizan continuamente las tecnologías de información geográfica automatizada más avanzadas [3]. Su interfaz se puede ver en la Figura 1.5.



Figura 1.5: Interfaz de la plataforma SIGPAC del Gobierno de España,

Entre los objetivos más importantes del Proyecto SIGPAC figuran los siguientes:

- **Facilitar a los agricultores la presentación de solicitudes**, mediante la producción de los soportes gráficos necesarios para las declaraciones de superficie.
- **Facilitar los controles administrativos**, ya que, la información digital ayudará a la Administración a identificar mejor el origen de los errores derivados de las declaraciones de los agricultores o de la grabación de los datos, y servirá de soporte documental para la resolución de casos dudosos detectados como resultado de estos controles.
- **Facilitar los controles sobre el terreno**, agilizando la localización de parcelas y permitiendo la realización de "visitas rápidas" tanto en los controles clásicos como de teledetección.

Índices de Vegetación en el Cultivo del Olivar

En el contexto agrícola, la reflectancia se convierte en una herramienta esencial para comprender la salud de los cultivos, incluido el olivar. La reflectancia es la medida de la cantidad de luz y calor que una superficie o color refleja o absorbe. Este fenómeno revela información importante sobre el estado fisiológico de las plantas y sus circunstancias ambientales al medir la cantidad de luz reflejada por su superficie.

Los índices de vegetación se derivan de la capacidad de los cultivos para absorber y reflejar la luz en diferentes longitudes de onda. Estos índices son una forma importante de evaluar el vigor y la vitalidad de los olivos.

El espectro electromagnético

Las ondas electromagnéticas son emitidas por una variedad de fuentes naturales y artificiales. Esas ondas están formadas por campos eléctricos y magnéticos oscilantes que interactúan recíprocamente y de diversas formas con diferentes sistemas biológicos, como células, plantas, animales y humanos. Para entender mejor la relación mutua, es fundamental comprender las características físicas de las ondas que componen el espectro magnético [4].

Una onda electromagnética se caracteriza por:

- **Amplitud:** distancia entre el punto más alejado de ella y el de equilibrio o medio.
- **Longitud de onda:** distancia entre dos picos sucesivos de una onda.
- **Frecuencia:** número de veces que cambia el sentido del campo en la unidad de tiempo en un punto dado. Se mide en ciclos por segundo o hercios.

El espectro electromagnético, como se puede ver en la Figura 1.6 es el conjunto de longitudes de onda de todas las radiaciones electromagnéticas [5], por tanto, incluye:

- Los **rayos gamma** tienen las frecuencias más altas y las longitudes de onda más cortas. Las ondas de alta energía son las más penetrantes y pueden viajar a largas distancias a través del aire.
- Los **rayos X** tienen mayor energía que la radiación ultravioleta a pesar de tener longitudes de onda más largas que los rayos gamma. Los rayos gamma son emitidos por el núcleo, mientras que los rayos X son emitidos por electrones que están fuera del núcleo.
- La **radiación ultravioleta (UV)** se define como la porción del espectro electromagnético que se encuentra entre los rayos X y la luz visible.
- La **luz visible** es la parte de espectro electromagnético que los ojos humanos son capaces de detectar. Cubre todos los colores del azul a 400 nm al rojo a 700 nm.

- La **radiación infrarroja (IR)** es la parte del espectro electromagnético que se encuentra entre la luz visible y las microondas. La fuente natural más importante de radiación infrarroja es el Sol.
- Las **ondas radioeléctricas** tienen longitudes de onda largas que varían unos pocos centímetros a miles de kilómetros de longitud. Sus principales usos son en la televisión, los teléfonos móviles y las comunicaciones por radio. [5]

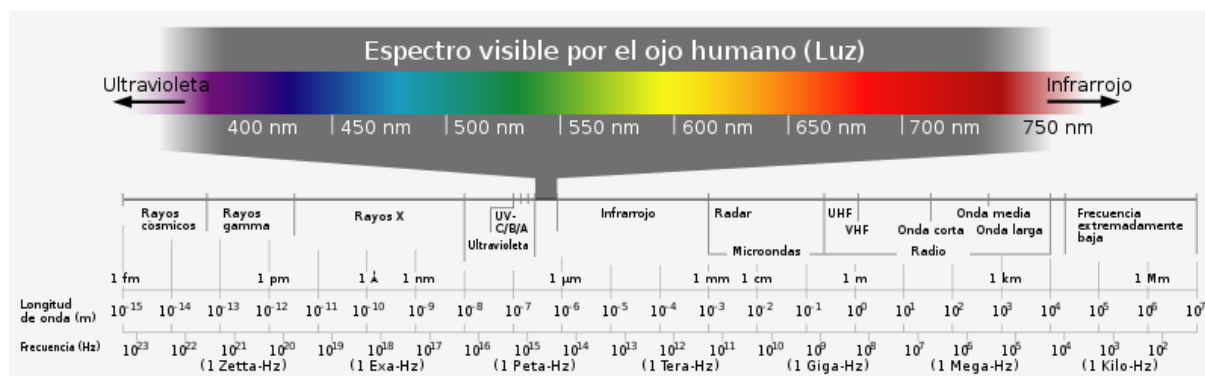


Figura 1.6: Espectro Electromagnético - Concepto

Para el cálculo de los índices de vegetación nos interesan desde el infrarrojo térmico hasta el espectro visible principalmente para el cálculo de los mismos ya que las cubiertas vegetales, en general, responden de maneras muy específicas en estos rangos que nos son útiles para saber su estado.

Índices de vegetación

Los índices de vegetación son el resultado de una función algebraica que utiliza una o varias bandas del espectro electromagnético, son imágenes que se han calculado utilizando operaciones algebraicas entre varias longitudes de espectro [6]. Por tanto, son obtenidos a partir de imágenes multispectrales y permiten evaluar el estado de salud de los cultivos a través de una variedad de características, como su contenido de clorofila, su estado nutricional, su estado hídrico y sus deficiencias estructurales [7].

Las cámaras que se utilizan se denominan cámaras multispectrales porque captan múltiples espectros. Se utilizan los colores verde, azul, rojo e infrarrojo cercano para índices de vegetación significativos. Se deben incluir en un programa cuando la cámara captura imágenes de estas bandas para manipular las diferentes fórmulas de los índices, obteniendo valores para poder actuar en consecuencia sobre nuestros cultivos.



Figura 1.7: Ejemplo de índices de vegetación - Geoinnova

Existen centenares de índices de vegetación que se han ido desarrollando a través de investigaciones en las últimas décadas, por ello, para nuestro estudio hemos seleccionado los siguientes por ser los más representativos para nuestro estudio:

- **NDVI (Normalized Difference Vegetation Index)**

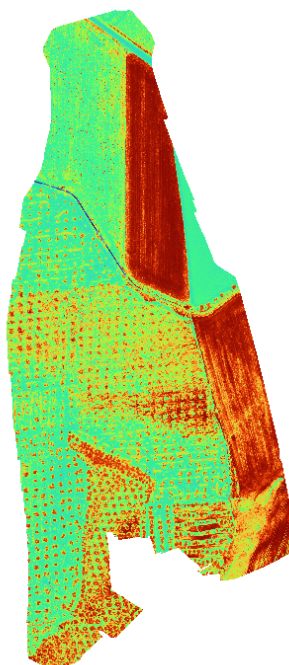


Figura 1.8: Ejemplo NDVI Pseudocolor Monobanda - Elaboración propia

El Índice de Vegetación de Diferencia Normalizada es el más utilizado [8]. Este índice mide la salud y el verdor de la vegetación. Es un indicador numérico que

utiliza las bandas espectrales roja e infrarrojo cercano, ya que aquellas áreas que reflejan más en el espectro del infrarrojo cercano corresponde a zonas con vegetación más densa y saludable. Se puede ver un ejemplo en la Figura 1.8.

Calcula por tanto la diferencia entre el infrarrojo cercano (NIR), ya que la vegetación se refleja fuertemente esta banda y la luz roja, ya que la vegetación tiene una reflectancia más baja en esta.

INTERPRETACIÓN NDVI

Este índice está definido por valores que van de -1.0 a 1.0, donde los valores negativos están formados principalmente por nubes, agua y nieve, y los valores negativos cercanos a cero están formados principalmente por rocas y suelo descubierto [9] [10].

- **[0.1 o menos]:** estos valores tan pequeños corresponden a áreas sin rocas, arena o nieve.
- **[0.2 - 0.3]:** estos valores moderados representan arbustos y praderas.
- **[0.6 - 0.8]:** los valores grandes indican bosques templados y tropicales.

$$NDVI = \frac{(NIR - RED)}{(NIR + RED)}$$

- **NIR:** reflectancia en la banda del infrarrojo cercano.
- **RED:** reflectancia en la banda del rojo.

■ SAVI (Soil Adjusted Vegetation Index)

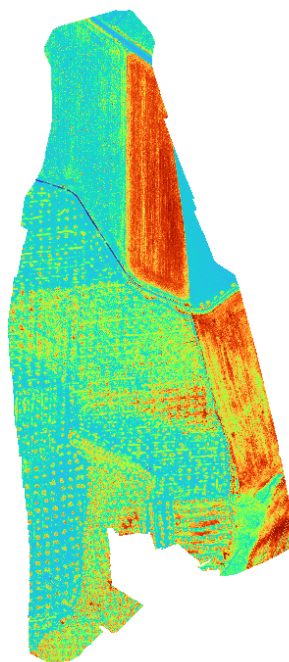


Figura 1.9: Ejemplo SAVI Pseudocolor Monobanda - Elaboración propia

Este índice es creado para reducir el impacto del brillo del suelo utilizando el coeficiente L en la ecuación del índice NDVI, descrito anteriormente. De esta manera, corrige el impacto que tiene en los estudios de vegetación. Cuando la vegetación se encuentra sobre suelos expuestos, este índice evita distorsiones en los valores de análisis [11]. Se puede ver un ejemplo en la Figura 1.9.

El factor L amortigua la presencia del suelo a través de valores de 0 a 1 para zonas con gran densidad vegetal. De esta manera, en suelos con desarrollo vegetativo, el factor L pasa a valor 0, lo que altera la ecuación y la hace equivalente al NDVI. El factor L pasa a valor 1, amortiguando el índice inicial NDVI cuando hay vegetación en desarrollo influenciada por la exposición del suelo. Los suelos medianamente expuestos tendrán por defecto un valor de factor L igual a 0.5.

Este índice es especialmente adecuado para estudios que analizan la vegetación en etapas tempranas de crecimiento o en áreas con vegetación dispersa. En general, el índice SAVI puede considerarse como una alternativa viable cuando se encuentran suelos con una baja densidad vegetal y la exposición de la superficie del suelo es relevante como ocurre en el olivar tradicional.

INTERPRETACIÓN SAVI [12]

- **[-1.5 - 0]:** presencia de agua ya que absorbe la mayoría de la radiación en las bandas espectrales utilizadas para calcular el índice.
- **[0.01 - 0.37]:** alta densidad de vegetación, los valores positivos bajos se refieren a áreas con vegetación saludable y densa.

- **[0.38 - 0.76]:** cobertura vegetal moderada, es decir, áreas con vegetación menos densa o zonas donde la vegetación está en crecimiento o no tan saludable.
- **[0.77 - 1.10]:** baja densidad de vegetación, puede ocurrir en áreas con vegetación escasa, en declive o estresada.
- **[1.11 - 1.50]:** ausencia de agua y muy poca o ninguna vegetación. Puede ocurrir en áreas urbanas, suelos desnudos o desiertos.

$$SAVI = \left(\frac{(NIR - RED)}{(NIR + RED + L)} \right) \cdot (1 + L)$$

NIR: reflectancia en la banda del infrarrojo cercano.

RED: reflectancia en la banda del rojo.

L: factor de corrección que varía según la densidad de la vegetación.

■ **NDMI (Normalized Difference Moisture Index)**

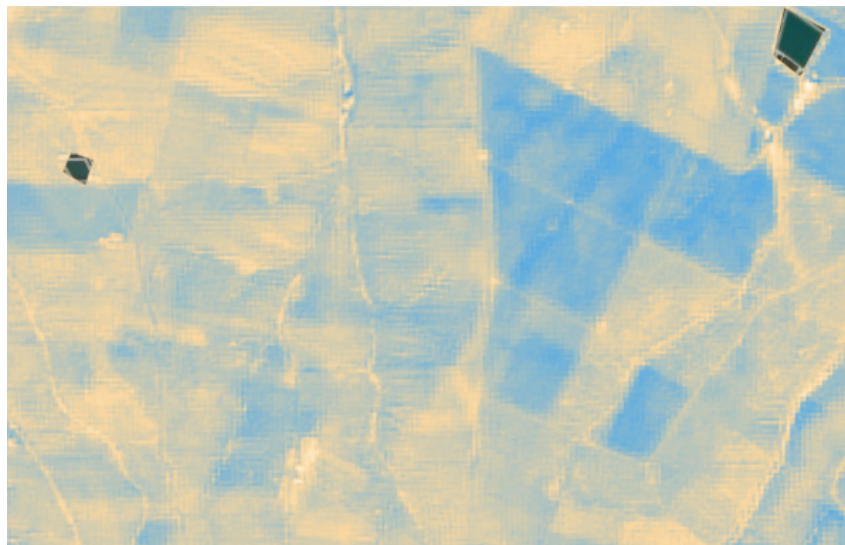


Figura 1.10: Ejemplo NDMI Invierno 2022 - Elaboración propia

Este índice caracteriza el nivel de estrés hídrico en plantas, es decir, detecta los niveles de humedad en la vegetación mediante una combinación de bandas espectrales de infrarrojo cercano (NIR) e infrarrojo de onda corta (SWIR). Se utiliza para monitorear el estrés por sequía y la vegetación. Los indicadores NDMI tienen una fuerte correlación con materia seca en hojas [13]. Se puede ver un ejemplo en la Figura 1.10.

Es un indicador fiable del estrés hídrico en los cultivos. Es muy importante ya que puede detectar condiciones severas de sequía que no solo estresan los cultivos

sino que pueden destruir todo el rendimiento. Los investigadores utilizan NDMI para monitorear los cambios en el contenido de agua de las hojas.

Debido a que la reflectancia de las plantas es ligeramente diferente en cada fase fenológica, es importante tener en cuenta que los valores del NDMI varían a lo largo de la temporada de crecimiento. Los valores del NDMI y un NDVI significativamente inferiores pueden confirmar el estrés hídrico [14].

INTERPRETACIÓN NDMI [14]

Sabiendo que el dosel vegetal se refiere a la capa superior de vegetación en un ecosistema, podemos clasificar los valores en:

- **[-1 – -0.8]:** estos valores indican suelo desnudo, no hay vegetación.
- **[-0.8 – -0.6]:** cobertura del dosel casi inexistente, la mayor parte del área es suelo desnudo.
- **[-0.6 – -0.4]:** cobertura del dosel muy baja, vegetación escasa.
- **[-0.4 – -0.2]:** cobertura del dosel baja y seca o muy baja y húmeda, puede representar áreas con vegetación seca o vegetación muy dispersa pero húmeda.
- **[-0.2 – 0]:** cobertura del dosel media-baja y estrés hídrico alto o cobertura del dosel baja y estrés hídrico bajo, áreas con vegetación dispersa con alta necesidad de agua o vegetación más uniforme pero no muy abundante con suficiente agua.
- **[0 – 0.2]:** cobertura media del dosel y alto estrés hídrico o cobertura media-baja del dosel y bajo estrés hídrico, vegetación moderada que puede estar sufriendo por falta de agua o vegetación menos densa pero con suficiente humedad.
- **[0.2 – 0.4]:** cobertura del dosel media-alta y estrés hídrico alto o cobertura media y estrés hídrico bajo, áreas con vegetación moderadamente densa que podría necesitar más agua o vegetación abundante con buen estado hídrico.
- **[0.4 – 0.6]:** cobertura del dosel alta, sin estrés hídrico, es decir, vegetación densa y saludable con suficiente agua.
- **[0.6 – 0.8]:** cobertura del dosel muy alta, sin estrés hídrico, vegetación muy densa en excelente estado.
- **[0.8 – 1]:** cobertura total del dosel, sin estrés hídrico/anegamiento, vegetación extremadamente densa y saludable, posiblemente en condiciones de anegamiento donde hay demasiada agua.

$$NDMI = \frac{(NIR - SWIR)}{(NIR + SWIR)}$$

NIR: reflectancia en la banda del infrarrojo cercano.

SWIR: reflectancia en la banda del infrarrojo de onda corta.

1.7.3. Imágenes Satelitales

Debido a que en este proyecto, se presenta un análisis de diversos índices de vegetación, así como de las lluvias y precipitaciones registradas a lo largo del tiempo, se debe indicar la procedencia de estos datos. Esta ha sido de tres fuentes satelitales principales: Sentinel-2, MODIS LST y ERA5-Land. En los siguientes apartados, se detalla el uso de estos satélites.

Sentinel-2

Sentinel-2 es parte del Programa Copernicus de la Agencia Espacial Europea (ESA). Se lanzó para proporcionar datos satelitales de alta resolución para el monitoreo del uso y cobertura del suelo, el cambio climático y la gestión de desastres. La constelación Sentinel-2 está compuesta por dos satélites gemelos, Sentinel-2A y Sentinel-2B, lanzados en 2015 y 2017 respectivamente. Estos satélites están equipados con instrumentos de imagen multiespectral (MSI) que registran 13 bandas de anchos amplios [15].

Este satélite, revisa cada cinco días las zonas manteniendo los mismos ángulos de visión, menos en latitudes altas que tiene diferentes. La resolución espacial es de 10 metros, 20 metros y 60 metros y su campo de visión es de 290 kilómetros. Podemos observar una fotografía de este en la Figura 1.11.

Los datos de Sentinel-2 son esenciales para monitorear cambios rápidos en la superficie terrestre, impulsados por la expansión agrícola, el cambio climático y la urbanización. Estos datos ayudan a evaluar el estado de la superficie terrestre y a informar a los responsables de la toma de decisiones sobre los cambios futuros, contribuyendo también al seguimiento de los Objetivos de Desarrollo Sostenible (ODS).

El monitoreo a largo plazo por Sentinel-2 fortalece las políticas existentes al proporcionar información precisa y oportuna, y ha sido crucial para desarrollar mapas de

alta resolución espacial.



Figura 1.11: Sentinel-2 - European Space Agency

MODIS LST

MODIS (Moderate Resolution Imaging Spectroradiometer) es un instrumento clave a bordo de los satélites Terra y Aqua de la NASA, lanzados en 1999 y 2002 respectivamente.

MODIS observa la Tierra diariamente con 36 bandas espectrales, abarcando desde el visible hasta el infrarrojo térmico. Proporciona datos críticos para el estudio del cambio climático, la gestión de recursos naturales, y la monitorización ambiental. Entre sus aplicaciones destacan la medición de la Temperatura de la Superficie Terrestre (LST), la evaluación de la cubierta terrestre, la detección de incendios, y el seguimiento de la calidad del aire [16]. Podemos ver una imagen de este en la Figura 1.12.

Dentro de sus aplicaciones, debemos destacar el producto de Temperatura de la Superficie Terrestre (LST). Es una herramienta crucial para estimar la temperatura del aire (T_{air}) en diversas aplicaciones ambientales. Este sensor proporciona datos de LST que son esenciales para entender y monitorear el clima, los procesos biológicos terrestres y la hidrología a escalas regionales y globales [17].

La resolución espacial del satélite, que capta datos en 36 franjas espectrales, tiene una resolución espacial de 250 metros en las bandas 1-2, 500 metros en las bandas 3-7 y 1 kilómetro en las bandas 8-36.



Figura 1.12: MODIS - NASA Ocean Color

ERA5-Land

ERA5-Land es un conjunto de datos globales mejorados para la componente terrestre del quinto ciclo de reanálisis europeo (ERA5), producido por el Centro Europeo de Previsiones Meteorológicas a Medio Plazo (ECMWF) dentro del Servicio de Cambio Climático de Copernicus (C3S). Este dataset cubre el periodo desde 1950 hasta el presente. Podemos ver una imagen de este satélite en la Figura [1.13](#).

ERA5-Land describe la evolución de los ciclos de agua y energía sobre la tierra. La resolución espacial nativa de ERA5-Land era de 0.1 grados (9 km), actualmente se ha mejorado a 30 segundos de arco (1 km). Comparado con ERA5 y el antiguo ERA-Interim, ERA5-Land ofrece una mejor resolución espacial y temporal, lo que mejora la descripción del ciclo hidrológico, especialmente en la humedad del suelo y las estimaciones de descarga de ríos.

La alta resolución espacial y temporal de ERA5-Land, junto con su consistencia a lo largo del tiempo, lo convierten en un recurso valioso para estudios hidrológicos, la inicialización de modelos de predicción meteorológica y climática, y aplicaciones en la gestión de recursos hídricos y medioambientales [[18](#)].

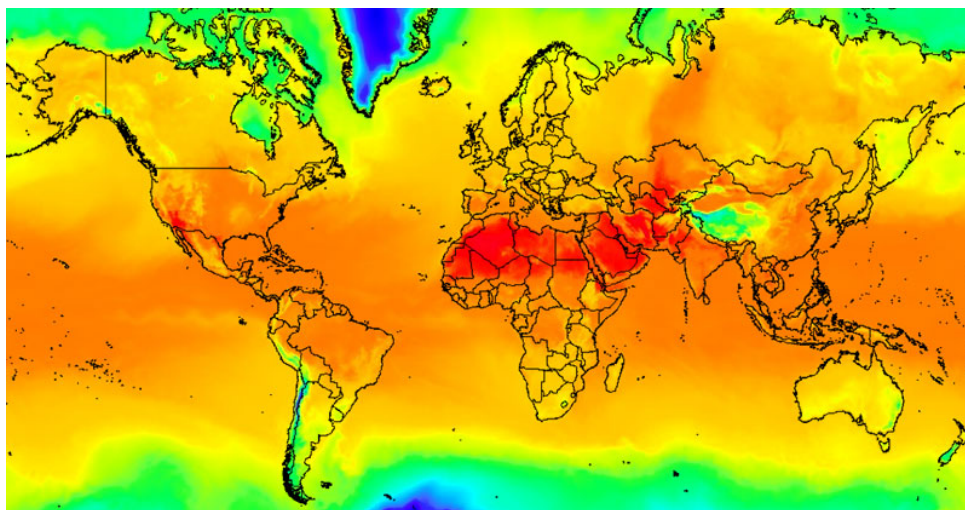


Figura 1.13: Vista ERA5-Land

1.7.4. Enfermedades del olivar

El objetivo de este trabajo es mejorar la salud del olivar, por ello, una gran fuente de pérdidas de la cosecha del olivar se trata de las enfermedades o plagas que le puedan ocurrir. Hay muchas de ellas pero, tras consultar con varios técnicos agrícolas, se seleccionaron para su estudio las cinco más frecuentes:

Euzophera Pinguis



Figura 1.14: Euzophera Pinguis - Process Olive

NOMBRE COMÚN: abichado del olivo.

DESCRIPCIÓN: se trataba de una enfermedad secundaria pero en los últimos años ha tomado mucha importancia [19]. El adulto es una mariposa de 2-2,5 cm con alas de color marrón con dos bandas transversales más claras. Deposita unos huevos ovalados, aplanados y reticulados, de color blanquecino que posteriormente viran al rosa, difíciles de ver (1x0,75 mm). La larva llega a alcanzar los 2,5 cm [20].

CICLO BIOLÓGICO: en Andalucía occidental presenta dos generaciones pero es frecuente encontrar los estados de desarrollo durante todo el año (excepción invierno).

- **1ª GENERACIÓN:** en febrero empiezan a verse las primeras crisálidas y según los años empiezan a volar los adultos, en abril se produce el máximo vuelo de esta generación. En este mes aparecen las primera larvas pequeñas, su presencia es máxima a mediados de mayo.
- **2ª GENERACIÓN:** A principios de Septiembre aumenta la crisalidación cuya puesta finaliza a mediados de octubre. Menos intensa que en primavera.

Los huevos son depositados cerca de grietas o heridas de poda, y la larva recién nacida penetra en el interior del árbol haciendo una galería entre corteza y madera de la que se alimentará toda su vida.

DAÑOS: galerías subcorticales de alimentación de las larvas. Las ramas o el tronco muestran un aspecto deprimido, amarillean y pueden secarse. Los ataques más graves se corresponden a una profusión de heridas consecuencia de heladas o de granizadas.

ESTRATEGIA DE LUCHA: control muy difícil. Las larvas recién nacidas son más sensibles y accesibles al control químico.

- Mezcla de un organofosforado y aceite de verano en abril-mayo y octubre. Evitando días calurosos, mínima presión y mojando abundantemente el tronco y las ramas principales, no la copa.
- Provocar las mínimas heridas posibles en periodos de máximo vuelo de adultos, evitar golpes de sol en troncos, vigorizar los árboles tras heladas o granizadas.

Bactrocera oleae Gmel

Figura 1.15: Bactrocera oleae Gmel - Junta de Andalucía

NOMBRE COMÚN: mosca del olivo.

DESCRIPCIÓN: es la plaga que más perjuicio económico produce en el cultivo del olivo, su incidencia es muy variable en cada campaña. El individuo es una mosca típica de colores marrones. Las hembras son mayores que los machos. El huevo es blanco, alargado y cilíndrico. Son depositados bajo la epidermis de la aceituna en una pequeña cámara que la hembra prepara con su oviscapto. Solo se aprecia un pequeño corte y una manchita [21].

CICLO BIOLÓGICO: los adultos vuelan casi todo el año, bajando sus poblaciones hasta casi desaparecer en abril y mayo. A partir de junio, coincidiendo con periodos de temperaturas suaves y lluviosas, se inicia la puesta en la aceituna. Sufren una alta mortandad debido a las altas temperaturas y baja humedad ambiental. En otoño se activa de forma notable y se aumenta progresivamente los índices de aceituna picada.

DAÑOS: en aceitunas de mesa la destrucción de tejidos durante su evolución larvaria, desprecian el fruto y en aceituna para aceite hay daños directos, pérdida de peso y caída prematura del fruto.

ESTRATEGIA DE LUCHA: para medir los niveles de población de adultos pueden emplearse placas amarillas engomadas con feromonas o mosquero con atrayente alimenticio (fosfato biamónico al 4%). En las hembras capturadas podemos medir el número de huevos, con lo que obtendríamos el índice de fertilidad y de riesgo. Todo ello junto con el porcentaje de aceitunas picadas, indicador fundamental, nos orienta sobre el momento idóneo de intervención.

Para combatir adultos: mezcla de insecticida y atrayente alimenticio o sexual, aplicado en bandas o parcheo. Para combatir larvas en frutos: insecticidas organofosforados de gran poder penetrante. Los factores naturales de nuestro verano, llegan a ser en algunos casos muy limitantes [22].

Spilocea Oleagina

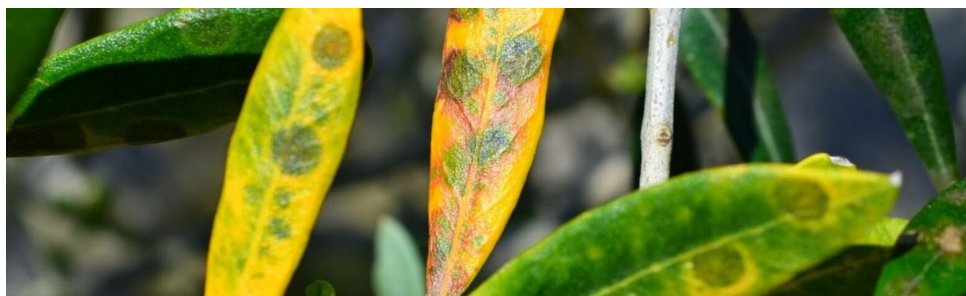


Figura 1.16: Spilocea Oleagina - Probelte

NOMBRE COMÚN: repilo

DESCRIPCIÓN: enfermedad más grave del olivar, está presente en todas las áreas del mundo donde se cultiva el olivo, aunque su distribución en olivares está determinada por la susceptibilidad de la variedad o el genotipo, la virulencia del patógeno y las condiciones ambientales, sobre todo las que favorezcan una elevada humectación foliar [23].

CICLO BIOLÓGICO: el patógeno sobrevive el verano en las hojas enfermas en forma de esporas, pegadas a las estructuras reproductivas del hongo. También quedan debajo de la cutícula de las hojas sus estructuras vegetativas que se reactivan al llegar las lluvias de otoño y dan nuevas generaciones de esporas. Estas se dispersan dentro de cada árbol por lavado de la lluvia y el rocío.

Para los procesos de germinación se necesita agua libre o humedad de saturación durante 1-2 días según la temperatura (5-25º con óptimo de 12º). El período entre la penetración y la aparición de síntomas dura de 4-15 semanas según la humedad, temperatura, edad de la hoja y variedad.

En nuestras condiciones puede haber hasta cuatro ciclos de enfermedad en un año. Si llueve al final de la primavera se producen las últimas infecciones que quedan latentes durante el verano y se reactivan al final del verano, reanudando los ciclos de la enfermedad en otoño-invierno.

DAÑOS: dependen de las condiciones ambientales y son diferentes según la época del año en la que se desarrollan.

- **Hojas:** en el haz se producen manchas circulares de color marrón oscuro debido a las fructificaciones (esporas) del hongo. En primavera las manchas desarrollan un halo amarillo. Al envejecer las lesiones evolucionan dando anillos concéntricos amarillos, marrones y verdes (“ojos de pavo real”). En verano pueden aparecer manchas plateadas correspondientes a lesiones primaverales cuyo desarrollo se detuvo antes de esporular por temperaturas altas.
- **Frutos:** son raros y consisten en manchas deprimidas de color pardo-negruzco de forma más o menos circular y tamaño variable. Se hace más patente a medida que se produce la maduración de la aceituna ya que la zona colonizada por el hongo se mantiene verde pero el resto del fruto negro. Hay caída de aceituna y merma la cosecha.

ESTRATEGIA DE LUCHA: deben elegirse las más resistentes para zonas húmedas. Aumentar la aireación de la copa. Aplicaciones de compuestos cúpricos al final del invierno y del verano y a final de la primavera mezclando el cobre con un fungicida sistémico para controlar infecciones tardías.

Prays oleae Bern



Figura 1.17: Prays oleae Bern - Agrotrápiche

NOMBRE COMÚN: prais del olivo.

DESCRIPCIÓN: Su principal huésped es el olivo. Tiene tres generaciones anuales en tres órganos vegetativos diferentes: flor, fruto y hoja. Provocan pérdidas considerables. El adulto es una polilla gris plateada. El huevo es blanquecino y amarillento al eclosionar. La crisálida la realiza protegida por sedas y restos vegetales.

CICLO BIOLÓGICO: hay tres generaciones y están sincronizadas con la evolución fenológica del olivo [24].

- **Generación filófaga:** los adultos en octubre y noviembre depositan sus huevos en las hojas y las larvas recién nacidas se mantienen en galerías interiores durante el invierno. En febrero aumenta su actividad, cambia varias veces de hoja y se alimentan exteriormente de yemas y hojas.
- **Generación antófaga:** en abril y mayo los adultos provenientes de la generación anterior depositan los huevos en botones florales cerrados y con marcada preferencia sobre el cáliz. Las larvas penetran dentro del botón y se alimentan de las anteras y el estigma. Esta generación es de evolución más rápida completándose en un mes y medio.
- **Generación carpófaga:** los adultos aparecen de mayo a junio y realizan la puesta en el cáliz. Cuando nacen las larvas perforan el fruto y entran en la almendra antes de que se endurezca el hueso. Se alimentan de la semilla hasta que a mediados de septiembre inicia la salida de la aceituna para crisalidar en el suelo.

DAÑOS: los de la primera generación son inapreciables. La segunda debido a su preferencia alimentaria sólo incide en la futura cosecha en casos de nivel de floración medio-bajo. La última produce la primera caída de frutos pequeños que funciona como aclareo. El daño más grave es a partir de septiembre, cuando la larva al salir del fruto hace que éste caiga al suelo.

ESTRATEGIA DE LUCHA: se puede actuar en la generación antófaga antes de las primeras crisálidas o en la carpófaga con productos de alto poder penetrante. La eficacia aumenta conforme más y mejor se mojen los frutos [25].

Verticillium dahliae Kleb

Figura 1.18: Verticillium dahliae Kleb - ABC Sevilla

NOMBRE COMÚN: verticilosis

DESCRIPCIÓN: enfermedad causada por un hongo. La fuente principal de infecciones son los microesclerocios que están en el suelo, donde pueden persistir más de diez años. La mayoría están en la capa arable aunque pueden hallarse a un metro de profundidad [26].

CICLO BIOLÓGICO: la incidencia es mayor en regadío que en secano y temperaturas del aire de 20-25° C (máximo de 30-35°) favorecen su desarrollo.

DAÑOS: apoplejía: desarrollo rápido y se manifiesta en otoño e invierno. Se produce una seca rápida de brotes y ramas que puede ocasionar hasta la muerte del árbol. Cuando hay un decaimiento lento, en primavera principalmente, aparecen necrosis y momificado de las inflorescencias y caídas de hojas excepto las del extremo del ramo. Los árboles jóvenes pueden morir pero los viejos solo se ven afectados algunos sectores.

ESTRATEGIA DE LUCHA: usar plantones libres del patógeno, ponerlos en suelos no infectados y evitar la introducción del hongo con las labores. Se deben eliminar los tejidos infectados, no intercalar siembras de cultivo susceptibles, destruir las malas hierbas, fertilización equilibrada en N y K y manejo adecuado del riego.

Tabla de Enfermedades

Tras presentar todas las enfermedades más comunes, se ha elaborado una tabla común con todas ellas, agrupando en las categorías NULO, BAJO, MEDIO, ALTO y EXTREMO los factores que pueden hacer evidentes la aparición de la enfermedad, se puede ver en la Figura 1.19.

Para hacerla lo más acertada posible, se ha realizado en base a todas las referencias mostradas en los apartados anteriores. Además, se ha revisado con la ayuda de Juan Diego Melero Blanca, experto en el tema y gerente de Agroquímicos Calatrava.

Los factores que intervienen son los siguientes:

- **Temperatura:** temperatura que debe hacer para que las diferentes enfermedades lleguen a prosperar, se eligió para el estudio en la Iteración Primera 4.3.
- **Humedad:** humedad necesaria para la prospección de las diferentes enfermedades, se han tomado los factores precipitaciones junto con el índice de vegetación SAVI.
- **Presencia/ausencia de depredadores:** si hay depredadores es más difícil que la enfermedad se de.
- **Fuera de la época / época:** si está fuera de la época, es muy difícil que la enfermedad aparezca, si está, se indica la época de esta.
- **Presencia/ausencia de frutos:** hay enfermedades que atacan cuando hay frutos, por ello, es un factor bastante importante a considerar.
- **Precipitación:** si ha habido precipitaciones en los últimos años, también es un factor muy importante a considerar, se eligió para el estudio en la Iteración Primera 4.3. **Inóculo del patógeno:** se refiere a la cantidad de patógenos presentes en un medio o en una superficie que tienen la capacidad de causar una infección o enfermedad en un organismo.

ENFERMEDAD	NOMBRE CIENTÍFICO	NULO	BAJO	MEDIO	ALTO	EXTREMO
Abichado del olivo	Euzophera Pinguis	-Temperatura: inferior a 10°C o superior a 35°C. -Humedad: inferior al 40% -Presencia de depredadores -Fuera de la época	-Temperatura: entre 10°C y 20°C -Humedad: entre 40% y 60% -Presencia de depredadores -Fuera de la época	-Temperatura: entre 25°C y 30°C -Humedad: entre 40% y 60% -Presencia de depredadores -Época: abril-junio, agosto-octubre	-Temperatura: entre 25°C y 30°C -Humedad: superior al 80% -Ausencia de depredadores -Época: abril-junio, agosto-octubre	-Presencia de larvas -Temperatura: entre 25°C y 30°C -Humedad: superior al 80% -Ausencia de depredadores -Época: abril-junio, agosto-octubre
Mosca del olivo	Bactrocera oleae Gmel	-Fuera de la época -Ausencia de frutos -Temperatura: por encima de 35°C -Humedad: por debajo del 40% durante períodos prolongados.	-Temprano o tarde (julio-agosto octubre-noviembre) -Presencia de frutos: aceitunas verdes jóvenes -Temperatura: medias diarias por debajo de 20°C. Humedad: 40-60%	-Temporada: agosto-septiembre -Presencia de frutos maduros -Temperatura: medias diarias entre 20-30°C -Humedad: 60-80%	-Temporada alta con olas de calor (+ de 3 días con 30°) -Presencia de frutos abundantes. -Temperatura: 25-30° -Humedad: niveles superiores al 80%.	-Presencia de larvas -Temporada alta con olas de calor (+ de 3 días con 30°) -Presencia de frutos abundantes. -Temperatura: 25-30° -Humedad: niveles superiores al 80%.
Repilo	Spilocea Oleagina	-Fuera de la época. -Temperatura: <10°C o > 32°C. -Humedad: < 40%.	-Época: Invierno o primavera -Temperatura: 10°C-15°C. -Humedad: 40%-60%.	-Época: Invierno o primavera. -Temperatura: 15°C-20°C. -Humedad: 60%-80%.	-Época: Invierno o primavera. -Temporada alta: olas de calor. -Presencia de frutos. -Temperatura: 25°C-30°C. -Humedad: > al 80%.	-Epidemia en curso. -Época: Invierno o primavera. -Temporada alta: olas de calor. -Presencia de frutos. -Temperatura: 25°C-30°C. -Humedad: > al 80%.
Prais del olivo	Prays oleae Bern	-Época: fuera de la temporada alta. -Temperatura: <10°C o > 40°C. -Humedad: < 40%. -Ausencia de frutos.	-Época: Generación: Filófaga (invierno) o antófaga (primavera). -Temperatura: 10°C-15°C. -Humedad: 40%-60%. -Presencia de frutos limitada.	-Época: Antófaga (primavera) o carpófaga (verano). -Temperatura: 15°C-25°C. -Humedad: 60%-80%. -Presencia de frutos jóvenes.	-Temporada alta: primavera y verano. -Presencia de frutos maduros. -Temperatura: 25°C-35°C. -Humedad: > 80% -Generación: Carpófaga (verano) con condiciones climáticas ideales.	-Epidemia en curso -Temporada alta: primavera y verano. -Presencia de frutos maduros. -Temperatura: 25°C-35°C. -Humedad: > 80% -Generación: Carpófaga (verano) con condiciones climáticas ideales.
Verticilosis	Verticillium dahliae Kleb	-Inóculo del patógeno: ausente -Temperatura: <10°C o > 40°C -Precipitación: Anual < 200 mm -Humedad: < 40% -Época: Invierno	-Inóculo del patógeno: Niveles muy bajos (< 1 ppg) -Temperatura: 15°C - 22°C -Precipitación: Anual 400-600 mm -Humedad: 40%-60% -Época: Cualquier época	-Inóculo del patógeno: Niveles bajos (1- 10 ppg) -Temperatura: 10°C - 25°C -Precipitación: Anual 600-800 mm -Humedad: 40% - 60% -Época: Cualquier época	-Inóculo del patógeno: Niveles moderados (entre 10 y 40 ppg) -Temperatura: 20°C - 30°C -Precipitación: Anual entre 800 y 1000 mm -Humedad: 60% - 80% -Época: Primavera, Verano	-Inóculo del patógeno: Niveles altos (> 40 ppg) -Temperatura: > 25°C -Precipitación: Anual > 1000 mm -Humedad: > 80% -Época: Primavera, Verano

Figura 1.19: Tabla de riesgos de enfermedades - Elaboración Propia

Una vez que se ha introducido el proyecto, explicando en detalle sus objetivos, contexto, antecedentes y el estado del arte, procederemos a abordar la planificación detallada del mismo en el siguiente capítulo.

Capítulo 2

Planificación

En este apartado se presentará toda la planificación del proyecto, empezando por la decisión de la metodología de desarrollo a utilizar, la identificación de tareas de las diversas iteraciones, la planificación temporal con su correspondiente diagrama de Gantt y la estimación de costes y viabilidad donde agrupamos el modelo COCOMO y el presupuesto.

2.1. Metodología de desarrollo

En las últimas décadas, el desarrollo de software ha tenido una evolución bastante importante. Tradicionalmente, a la hora de plantear un proyecto software, se usaba el modelo de desarrollo en cascada, también conocido como ciclo de vida clásico (CVC).

Este modelo presentaba diversos inconvenientes ya que tenía una estructura lineal, como se puede ver en la Figura 2.1, que no permitía adaptarse bien a los cambios en los requisitos, por ello, la planificación era imprecisa ya que los requisitos son cambiantes. Con el objetivo de superar estas limitaciones y adaptarse a las necesidades cambiantes del mercado, han surgido nuevos paradigmas de desarrollo de software, entre ellos, se encuentran las metodologías ágiles [27].

Las metodologías ágiles surgieron como respuesta a las metodologías tradicionales que presentaban pesadez, inflexibilidad a los cambios y exceso en la producción de documentos. Estas metodologías fomentan el trabajo en equipo y mantienen una comunicación fluida entre el grupo de desarrollo y los clientes [28].

Se desarrollaron para adaptarse a los cambios en los requisitos, mejorar la gestión

del ciclo de vida del desarrollo de programas y ayudar a reducir y resolver los errores en cada etapa de desarrollo. Entre las metodologías para el desarrollo web podemos encontrar a Scrum o eXtreme Programming (XP).

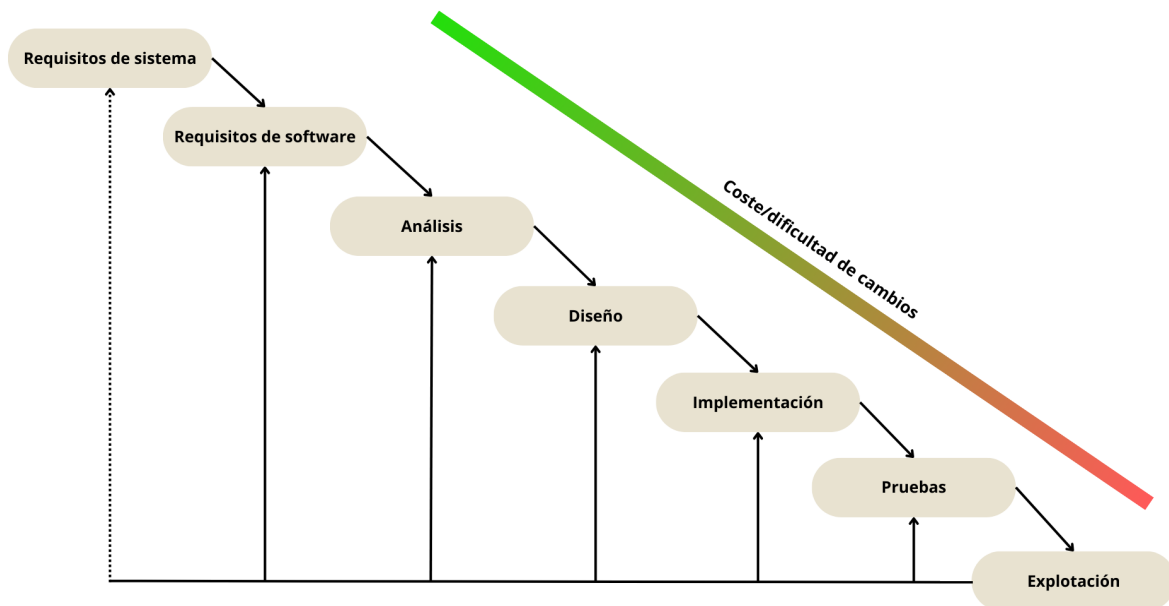


Figura 2.1: Modelo CVC - Elaboración propia

2.1.1. Scrum

Según la guía oficial de Scrum (Fig 2.2, este es un marco de trabajo a través del cual las personas pueden abordar problemas complejos adaptativos, a la vez que se entregan productos de forma eficiente y creativa con el máximo valor [29].

El problema que radica con esta metodología ágil es que sus roles, eventos, artefactos y reglas son inmutables y aunque es posible implementar solo partes de Scrum, el resultado no es este. El equipo de Scrum debe estar compuesto por al menos tres personas hasta un máximo de nueve, donde encontraremos los roles de Product Owner, Scrum Master y Equipo de Desarrollo.

Dentro de Scrum debemos destacar los siguientes términos:

- **Sprints:** son el corazón de Scrum, donde se convierten ideas en valor en eventos de hasta un mes. Incluyen planificación, reuniones diarias, revisión y retrospectiva. Su duración es de una a cuatro semanas.

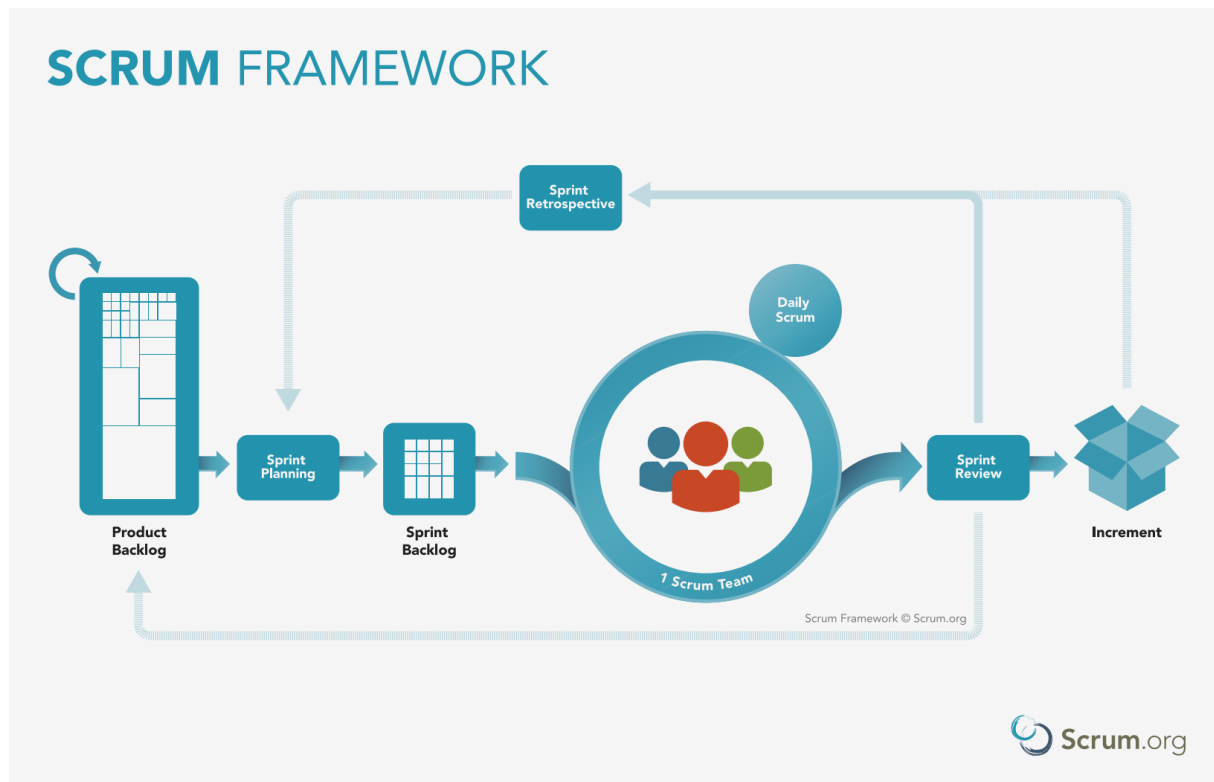


Figura 2.2: Scrum Framework

- **Sprint Planning:** la planificación del sprint inicia el trabajo del sprint mediante la colaboración del equipo Scrum. Se discuten y seleccionan ítems del Product Backlog, se define el objetivo del Sprint y se planifica cómo se completarán los ítems seleccionados.
- **Daily Scrum:** inspecciona el progreso hacia el objetivo del sprint y ajusta el Sprint Backlog en un evento de 15 minutos, mejorando la comunicación y auto-gestión del equipo.
- **Sprint Review:** inspecciona los resultados del Sprint y determina futuras adaptaciones. El equipo Scrum presenta su trabajo a los stakeholders y discuten el progreso hacia el objetivo del producto.
- **Sprint Retrospective:** planifica mejoras en calidad y efectividad. El equipo Scrum revisa el sprint pasado, identifica problemas y sugiere cambios para el próximo sprint.

2.1.2. Método iterativo/incremental

Debido a que este proyecto está realizado únicamente por una persona, no podríamos elegir a Scrum como metodología ágil ya que sería incorrecto, pero sí podemos tomarlo como referencia en ciertos aspectos. La metodología elegida es, por tanto, un método iterativo/incremental.

Cada iteración se puede entender como un mini proyecto donde se repite un proceso de trabajo similar para proporcionar un resultado completo sobre producto final, de manera que se puedan obtener los beneficios del proyecto de forma incremental.

En cada iteración se evoluciona el producto a partir de los resultados completados en las iteraciones anteriores, añadiendo nuevos objetivos/requisitos o mejorando los que ya fueron completados [30].

2.2. Identificación de tareas

El proyecto se planificará en base a iteraciones, lo que nos permitirá identificar y organizar las tareas a desarrollar en cada una de ellas. A continuación, se describen las diversas iteraciones del proyecto.

2.2.1. Primera Iteración: Estudio y gestión de datos de entrada

Dentro de la primera iteración se identificaron las siguientes subtareas:

- **Recolección de Datos de Fincas:** identificación y registro de fincas con detalles relevantes como ubicación, área, y año de registro. Las zonas específicas dentro de las fincas se definieron según nombres comunes proporcionados por informantes.
- **Registro de Olivos:** catalogación de los olivos dentro de cada finca, incluyendo información sobre su ubicación, identificación, y características geométricas.
- **Análisis de Factores Clave:** selección de índices de vegetación (SAVI, NDMI, NDVI), temperatura y precipitaciones como factores esenciales para el análisis de la salud del olivar, obteniendo estos datos mediante Google Earth Engine.

- **Datos de Vuelos de Drones:** integración de datos proporcionados por el Grupo de Investigación derivados de vuelos de drones para un análisis más detallado.
- **Integración de Datos:** transformación y adaptación de los datos recolectados para su inserción en la base de datos mediante sentencias SQL.
- **Cálculo de Índices de Vegetación:** uso de Google Earth Engine para obtener índices de vegetación específicos para cada olivo basado en sus polígonos envolventes.
- **Procesamiento de Datos de Drones:** obtención y procesamiento de información de vuelos de drones siguiendo un procedimiento similar al utilizado para las fincas.

2.2.2. Segunda Iteración: Diseño de la base de datos

Esta iteración se realizó paralelamente a la anterior ya que fuimos perfilando la base de datos inicial en función a los datos que fuimos obteniendo. Por tanto, dentro de esta segunda iteración, teníamos estas subtareas:

- **Análisis de la Base de Datos:** analizar la BD sugerida por el grupo de investigación, identificar las clases necesarias y obtener sus ventajas e inconvenientes.
- **Confeccionar nuevas clases necesarias:** por ejemplo, una para representar históricos de información para poder almacenar información a nivel del Finca.
- **Introducir nuevos atributos:** después de analizar los factores del anterior apartado, introducir nuevos atributos dentro de las clases para poder almacenar los datos.
- **Agregar nueva información:** una vez obtenidos los datos del apartado anterior, introducir toda la nueva información en la base de datos.

Una vez concluidas estas dos primeras iteraciones tendríamos una base de datos completa y funcional.

2.2.3. Tercera Iteración : Arquitectura del sistema y pruebas

En esta iteración, ya se empezaría con el diseño del backend de la aplicación web, por ello, las subtareas serían:

- **Evaluación de Frameworks Backend:** Investigación y comparación de diferentes frameworks de backend, analizando sus ventajas y desventajas.
- **Selección de Framework:** Elección de un framework para la implementación basada en la investigación previa y planificación detallada.
- **Implementación de la Capa de Dominio:** Definición de clases y sistemas necesarios para la funcionalidad principal de la aplicación.
- **Pruebas de Funcionalidad:** Realización de pruebas, incluyendo tests de las funciones creadas para asegurar su correcto funcionamiento.
- **Implementación de la Capa de Persistencia:** Conexión de la base de datos al proyecto y definición de entidades y repositorios para interactuar con la base de datos Oracle.
- **Pruebas de Integración:** Verificación de la correcta obtención de datos de la base de datos mediante pruebas y tests.
- **Desarrollo del Servicio REST:** Creación de DTOs para las entidades y desarrollo de un servicio REST que declare todas las rutas necesarias.
- **Pruebas de las Peticiones REST:** Verificación del correcto funcionamiento de las peticiones mediante pruebas y tests.

2.2.4. Cuarta Iteración: Interfaz de Usuario

Una vez tenemos el Backend funcionando, debemos realizar la interfaz de usuario siguiendo estas subtareas:

- **Buscar información y ejemplos de distintos Frameworks de Frontend:** obteniendo sus ventajas y desventajas.
- **Elección de un Framework de implementación:** búsqueda de información sobre él y planificación.
- **Implementación de las vistas principales de la aplicación.**
- **Implementación de los diversos componentes de la aplicación.**

2.2.5. Quinta Iteración: Análisis de los datos

En esta iteración se perfeccionó la interfaz de usuario para realizar el análisis temporal de los datos. Las subtarefas que podemos encontrar son, por tanto:

- **Visualización de Datos de Cosecha por Finca:** mostrar gráfica de kilogramos de aceitunas recolectadas y kilogramos de aceite producido.
- **Exploración de Datos Históricos por Finca:** botones por año para acceder a gráficas de índices vegetales, precipitaciones y temperatura por mes.
- **Análisis de Datos por Olivo:** permitir selección de olivo y mostrar gráficas específicas de índices vegetales por año.

2.3. Planificación temporal

Una vez se han especificado todas las iteraciones junto con sus correspondientes tareas, debemos situarlas en la línea temporal. La realización de este proyecto se prevee que sea entre los meses de febrero a junio de 2024 de forma continua, es decir, todos los días de lunes a viernes se le dedicará un periodo de 4 horas. A continuación se va a especificar el periodo que se estimó para cada una de las iteraciones.

- **Tareas previas al proyecto:** 7 días. En esta etapa se deben perfilar los detalles y estudiar la viabilidad del trabajo a realizar antes de definir las iteraciones.
- **Iteración 1:** 23 días laborables y un mes en total. Es la iteración que se piensa que durará más, aunque parte de ella se debe hacer conjuntamente a la iteración 2 ya que conforme se van obteniendo los datos, se debe perfilar la base de datos e insertar en ella.
- **Iteración 2:** 15 días laborables y 3 semanas. Parte de la iteración, como mencioné antes, se solapará con la anterior.
- **Iteración 3:** 10 días/2 semanas, en esta iteración se implementará el Backend.
- **Iteración 4:** 20 días/4 semanas. Se realizará la Interfaz de Usuario.
- **Iteración 5:** Es la iteración más importante del proyecto pues se trata del resultado de todo lo que se hará previamente, dentro de la interfaz de usuario se muestran una serie de gráficos informativos. La previsión de la iteración es de 11 días laborables.

La previsión de las iteraciones no es muy equitativa ya que, por ejemplo, la primera tiene más trabajo que las otras. Esto depende de la complejidad de cada una y de los conocimientos previos que se tuviera. La iteración cuarta se prevee que dure menos, pues se tenían conocimiento sobre Spring Framework pero la implementación de la interfaz de usuario se estima en más tiempo porque no se tienen conocimientos previos de Vue Js. También hay que tener en cuenta si en esa época se tiene más carga de trabajo de otras asignaturas. También hay que destacar un descanso en Semana Santa durante las fechas 22 de marzo al 31 del mismo mes.

2.3.1. Diagrama de Gantt

A continuación se incluirá el Diagrama de Gantt en el que se puede ver más claro lo antes mencionado. Este tipo de diagrama muestran explícitamente la duración de las actividades e indican cuándo estamos realizando varias tareas a la vez, como es el caso de las dos primeras iteraciones. La longitud de cada nodo se usa para representar la duración de cada actividad.

La aplicación que se usó para planificar el proyecto fue [ClickUp](#). Esta es una plataforma de gestión de proyectos y tareas que ofrece una amplia gama de características para la colaboración en equipo, la planificación de proyectos, el seguimiento del tiempo y más. Es conocido por su flexibilidad y personalización, lo que permite a los equipos adaptar la herramienta a sus propios flujos de trabajo y necesidades específicas. Se ha usado en la planificación del proyecto para generar el Diagrama de Gantt. En la Figura 2.3 se puede ver el diagrama de Gantt final.

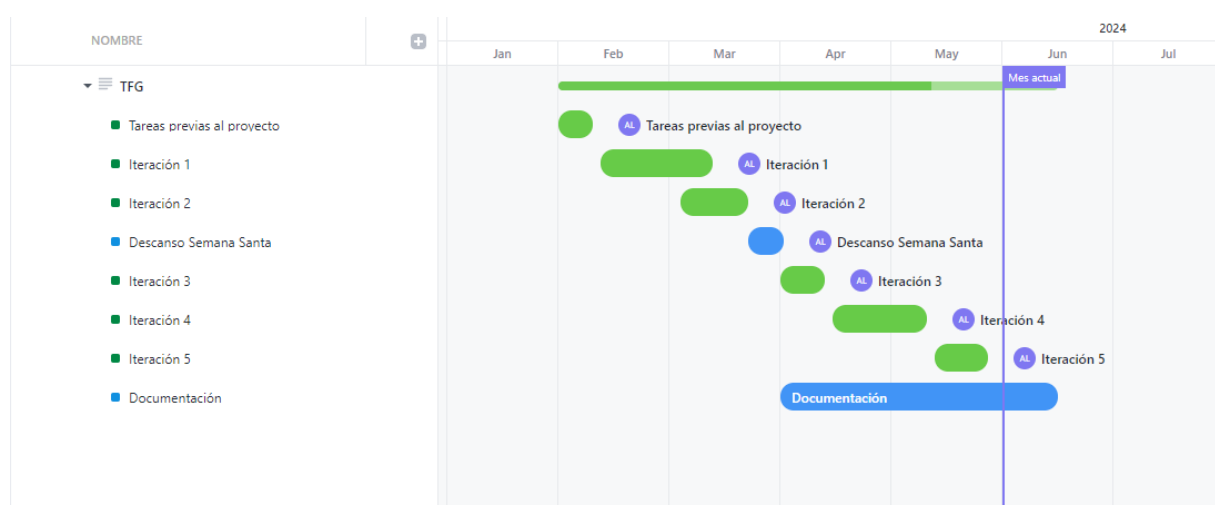


Figura 2.3: Diagrama de Gantt - Elaboración Propia

2.4. Estimación de costes y viabilidad

En este apartado, tras describir las distintas tareas a realizar junto con su planificación, debemos estimar el coste de los recursos y la viabilidad de estos ofreciendo un presupuesto final de este trabajo.

2.4.1. Estimación de tamaño y esfuerzo

Nuestro proyecto se trata de un Trabajo Fin de Grado, por tanto, no hay restricciones económicas. Sin embargo, si hay restricciones de tipo temporal ya que se compone de 12 créditos, lo que significan 300 horas aproximadamente. Los cálculos del tamaño del trabajo dependen, en gran parte, del tiempo disponible y el esfuerzo, por otro lado, están limitados a una persona, la que realiza este trabajo.

COCOMO

B. W. Boehm desarrolló el Modelo Constructivo de Costes a finales de los 70 y comienzos de los 80, y lo explicó detalladamente en su libro “Economía de la Ingeniería de Software” [31].

Se trata de un modelo que se basa en una estimación previa del tamaño del software en LDC (Líneas de Código), ya que es el parámetro básico para estimar el esfuerzo. Sus ecuaciones básicas son:

$$E = a \cdot (KLDC)^b$$

$$TD = c \cdot (E)^d$$

Medimos el esfuerzo (E) en personas-mes, TD mide el tiempo de desarrollo y a,b,c,d son parámetros según el tipo de desarrollo del proyecto.

Los tipos de desarrollo que existen en COCOMO son:

- **Orgánico:** Entorno de desarrollo estable con poca innovación técnica, tamaño pequeño (<50 KLDC) y tiempo de desarrollo corto. Adecuado para aplicaciones bien entendidas, donde los requisitos no cambian mucho. Comunicación informal

y estructuras jerárquicas flexibles, permitiendo mayor adaptabilidad y colaboración dentro del equipo.

- **Empotrado:** Software con requerimientos muy restrictivos, altamente volátil, complejo y de alta innovación técnica. Necesita planificación detallada, pruebas rigurosas y documentación formal. Integra profundamente con hardware y debe cumplir con estrictos estándares de rendimiento y fiabilidad. Requiere equipos de especialistas y un enfoque disciplinado.
- **Semi-Libre:** Intermedio entre orgánico y empotrado. Combina estabilidad e innovación moderada. Los proyectos varían en tamaño y complejidad. Estructura y comunicación más formales que en proyectos orgánicos, pero más flexibles que en empotrados. Necesita buena documentación y planificación, permitiendo adaptarse a cambios en los requisitos.

	Orgánico	Semiempotrado	Empotrado
Esfuerzo Estimado	$E_D = 2,4(KLDC)^{1,05}h-m$	$E_D = 3,0(KLDC)^{1,12}h-m$	$E_D = 3,6(KLDC)^{1,20}h-m$
Tiempo de Desarrollo	$T_D = 2,5(E_D)^{0,38}m$	$T_D = 2,5(E_D)^{0,35}m$	$T_D = 2,5(E_D)^{0,32}m$

Figura 2.4: Parámetros COCOMO - Elaboración propia

Además, hay tres versiones del modelo:

- **COCOMO Básico:** Estimaciones iniciales del proyecto que no son muy precisas. No se proporciona detalle sobre el proyecto. Utilizan la ecuación básica de COCOMO para calcular el esfuerzo y el tiempo de desarrollo.
- **COCOMO Intermedio:** Utiliza información adicional sobre el proyecto. La estimación del esfuerzo se adapta al entorno de desarrollo y se ajusta mediante 15 factores de coste distribuidos en 6 niveles.
- **COCOMO Detallado:** Proporciona detalle de cada componente independiente del sistema. Permite refinar y ajustar factores de coste, adaptándose a cada etapa del proyecto para obtener estimaciones más precisas y detalladas.

Para cada tipo de modelo COCOMO los valores constantes cambian como se ve en la Tabla 2.5. Además como nuestro modelo se corresponde con el COCOMO Intermedio, se muestran los 15 factores de coste distribuidos en 6 niveles, de Muy bajo a Extremadamente Alto 2.6.

Tipo	a	b	c	d
COCOMO básico				
Orgánico	2.40	1.05	2.50	0.38
Semilibre	3.00	1.12	2.50	0.35
Empotrado	3.60	1.12	2.50	0.32
COCOMO intermedio/detallado				
Orgánico	3.20	1.05	2.50	0.38
Semilibre	3.00	1.12	2.50	0.35
Empotrado	2.80	1.20	2.50	0.42

Figura 2.5: Valores constantes COCOMO - Elaboración propia

Una vez explicado el método, debemos aplicarlo a nuestro trabajo. En primer lugar, debemos seleccionar uno de los tres tipos de desarrollo que existen, podríamos clasificarlo como **semi-libre** ya que se encuentra en posición intermedia entre el orgánico y el empotrado pues es un entorno de desarrollo estable pero requiere cierta innovación. En cuanto a las versiones del modelo, lo podríamos clasificar como intermedio ya que debemos tener en cuenta los factores mencionados en la Figura 2.6 para un resultado más exacto.

Por tanto, teniendo en cuenta los factores mencionados tenemos:

- **Fiabilidad del software:** Nominal (1)
- **Tamaño de la base de datos:** Alto (1.08)
- **Complejidad proyecto:** Alto (1.15)
- **Restricciones de tiempo de ejecución:** Alto (1.30)
- **Restricciones de almacenamiento:** Nominal (1)
- **Volatilidad de la máquina virtual:** Nominal (1)
- **Tiempo de respuesta del equipo:** Alto (1.07)
- **Capacidad del analista:** Nominal (1)
- **Experiencia en la aplicación:** Bajo (1.13)
- **Capacidad de los programadores:** Alta (0.86)

Conductores de coste	Valoración					
	Muy bajo	Bajo	Nominal	Alto	Muy Alto	Ext. Alto
Fiabilidad del software	0.75	0.88	1	1.15	1.40	-
Tamaño de la base de datos	-	0.94	1	1.08	1.16	-
Complejidad proyecto	0.7	0.85	1	1.15	1.30	1.65
Restricciones de tiempo de ejecución	-	-	1	1.11	1.30	1.66
Restricciones de almacenamiento	-	-	1	1.06	1.21	1.56
Volatilidad de la máquina virtual	-	0.87	1	1.15	1.30	-
Tiempo de respuesta del equipo	-	0.87	1	1.07	1.15	-
Capacidad del analista	1.46	1.19	1	0.86	0.71	-
Experiencia en la aplicación	1.29	1.13	1	0.91	0.82	-
Capacidad de los programadores	1.42	1.17	1	0.86	0.7	-
Experiencia en el sistema operativo utilizado	1.21	1.10	1	0.90	-	-
Experiencia en el lenguaje de programación	1.14	1.07	1	0.95	-	-
Prácticas de programación modernas	1.24	1.10	1	0.91	0.82	-
Utilización de herramientas software	1.24	1.10	1	0.91	0.83	-
Limitaciones de planificación	1.23	1.08	1	1.04	1.10	-

Figura 2.6: Factores COCOMO - Elaboración propia

- **Experiencia en el sistema operativo utilizado:** Nominal (1)
- **Experiencia en el lenguaje de programación:** Nominal (1)
- **Prácticas de programación modernas:** Alta (0.91)

- **Utilización de herramientas software:** Alta (0.91)
- **Limitaciones de planificación:** Muy Alto (1.10)
- **Factor total:** 1.529329

Debemos calcular las líneas de código de nuestro proyecto:

- Líneas de código del programa de adaptación de datos: 2000 líneas
- Líneas de código del Backend: 3500 líneas
- Líneas de código del Frontend: 2500 líneas
- Líneas de código de la base de datos: 300 líneas
- TOTAL ESTIMADO: 8300 líneas de código

Ahora, debemos aplicar las fórmulas. Los coeficientes COCOMO asociados al modelo intermedio y semilibre son 3.0, 1.12, 2.5 y 0.35.

$$E = FactorCosteTotal * a \cdot (KLDC)^b$$

$$E = 1,529329 * 3 * (8,300)^{1,12} = 49,09 - > 50 Efectivos Por Mes$$

$$TD = c \cdot (E)^d$$

$$TD = 2,5 * (49,09)^{0,35} = 9,76 Meses$$

$$PersonalNecesario = 49,09/9,76 = 5,02 - > 5 Efectivos$$

Finalmente, tras la realización de los cálculos, vemos que se necesitarían 5 efectivos para un total de 10 meses, el proyecto ha durado 5 meses. Los resultados distan un poco de la realidad, ya que este trabajo solo lo ha realizado un efectivo al tratarse de un Trabajo Fin de Grado.

2.4.2. Presupuesto

En este apartado explicaremos el presupuesto del trabajo. El desarrollo de un proyecto genera una serie de costes. Aunque no hay un costo económico real al formar parte de un Trabajo de Fin de Grado, se realiza una estimación utilizando los diversos recursos utilizados en función de unas categorías:

1. **Hardware:** en este apartado incluimos el coste de los equipos informáticos usados para completar el proyecto, así como otros dispositivos que nos hayan ayudado a obtener información u otras consideraciones.
2. **Software:** se ha priorizado usar software gratuito que, en muchos casos, era por tener cuenta de estudiantes. En este apartado, debido a que hay que proporcionar la mejor solución, se hablarán de versiones no gratuitas para público común.
3. **Personal y servicios:** en este apartado nos referimos a costes asociados con la contratación de empleados necesarios para llevar a cabo las tareas del proyecto.
4. **Costes indirectos:** a parte de lo mencionado anteriormente, hay costes que son necesarios para el desarrollo de un proyecto, como instalaciones, facturas, etc., aunque no estén totalmente relacionados con el desarrollo del proyecto.

Además, se debe incluir la amortización de los bienes durante el mismo. Este es un método para distribuir el costo de un activo a lo largo de su vida útil, ya que no es posible deducir todo en el momento de la compra.

Los valores de amortización se derivan directamente de la tabla de coeficientes de amortización lineal de la Agencia Tributaria [32]. En cuanto a las fórmulas a usar, estas son:

$$BaseAmortizable = ValorInicial - ValorResidual$$

$$ValorResidual = \frac{ValorInicial}{VidaUtil}$$

$$AmortizacinAnual = BaseAmortizable * CoeficienteDeAmortizacion$$

$$AmortizacionProyecto = AmortizacionAnual * \frac{DuracionProyecto(Meses)}{12Meses}$$

Costes Hardware

El único coste hardware que ha influido en el proyecto es el equipo donde se ha desarrollado el mismo. A continuación se describen los costes del mismo:

- 2 Unidades Monitor Dell 24 – SE2422H = 215.38€
- Teclado multimedia Dell-KB216 - español (QWERTY) - negro = 24.38€
- Ratón óptico Dell MS116 (negro) = 15,67 €
- ASUS S501MD-7127000200 Intel Core i7-12700/16GB/512GB SSD = 894.98€
- **TOTAL= 1150.41€**

En cuanto a la amortización, según la tabla de coeficientes, para los equipos para procesos de información, su coeficiente lineal máximo es de 25 %, y su periodo máximo de 8 años. El resultado se puede ver en la Figura 2.7.

Origen	Vida útil (años)	Valor inicial (€)	Valor residual (€)	Amortización anual (€)	Amortización
Recursos Hardware	8	1150.41€	143.80€	251.65€	104.85€

Figura 2.7: Amortización del Hardware - Elaboración Propia

Costes Software

En este apartado se especificarán las distintas herramientas de software utilizadas para la realización del proyecto. La mayoría de ellas han sido gratuitas en su opción básica o debido a que tenían un plan para estudiantes. Así que, no se tendrán en cuenta las que sean gratuitas para el público general. La duración del proyecto ha sido de cinco meses así que se calcularán los costes en función a ello.

- Sistema operativo usado Windows 10 PRO = 168,99 €
- IntelliJ = 169.00 € al año
- Visual Paradigm = 32.20€ al mes

En cuanto a la amortización, según la tabla de coeficientes, para los sistemas y programas informáticos, su coeficiente lineal máximo es de 33 %, y su periodo máximo de 6 años. El resultado se puede ver en la Figura 2.8.

Origen	Vida útil (años)	Valor inicial (€ al año)	Valor residual (€)	Amortización anual (€)	Amortización del proyecto
Windows 10 Pro	6	168.99€	28.16€	56.33€	23.75€
IntelliJ	6	169.00 €	28.16€	56.33€	23.75€
Visual Paradigm	6	386.40€	64.40€	56.33€	53.52€
Total					100.97€

Figura 2.8: Amortizacion Software - Elaboración Propia

Costes Personal y Servicios

Al ser un Trabajo Fin de Grado, solo un desarrollador ha llevado a cabo todas las tareas del proyecto, por ello, para el presupuesto, solo debemos tener en cuenta un contrato de trabajo de una duración de cinco meses. Para obtener los datos necesarios para el presupuesto, debemos tener en cuenta el **XVIII Convenio colectivo estatal de empresas de consultoría, tecnologías de la información y estudios de mercado y de la opinión pública** [33].

Dentro del Boletín Oficial del Estado, el rol más adecuado para la realización del proyecto sería: **Área 3** (Desarrollo de software, Programación y Explotación de Sistemas), **Grupo C** (actividades de tipo técnico), **nivel 1** (cuenta con conocimientos necesarios).

Salario Base	Plus Convenio	Seguridad Social	Duración del proyecto
24.421,84€	2.112,28€	2892,97€	5 meses
Total		13.948,85€	

Figura 2.9: Coste Personal - Elaboración Propia

El salario base, desde enero de 2024, sería 24.421,84€ más el plus convenio de 2.112,28€. También tendríamos que añadir la Seguridad Social, según el régimen general, serían 2892,97€. Por tanto, el coste total aparece en la Figura 2.9

Costes Indirectos

En relación a los costes indirectos, estos gastos suelen agrupar las tarifas de luz, agua, conexión a Internet o lugar de trabajo. Estos gastos se estima que serán un sobrecosto del 10 % del presupuesto del proyecto final. Es difícil determinar a priori este tipo de gastos debido a su duración y la dificultad para determinar qué porcentaje de consumo real se consume durante los periodos de trabajo.

Coste Total del Trabajo Fin de Grado

Tras agrupar todos los costes explicados anteriormente tendríamos:

- Costes hardware=104.85€
- Costes software=100.97 €
- Costes personal y servicios=13.948,85€
- Costes Indirectos (10 %)= 1415.46€
- **TOTAL PROYECTO=15570.13€**

En este capítulo hemos realizado la planificación detallada del proyecto. En el siguiente capítulo, procederemos con el análisis general del software del sistema, definiendo el diagrama del dominio, casos de uso, diagramas de secuencia y perfiles de usuario.

Capítulo 3

Análisis del sistema

El análisis del sistema es una etapa fundamental en el desarrollo de software, ya que establece las bases sólidas sobre las que se fundamenta el proyecto. Un análisis completo y detallado permite garantizar el cumplimiento de los requisitos, optimizar el proceso de desarrollo, mejorar la calidad del software y reducir riesgos potenciales.

En este capítulo, se presentarán en detalle los aspectos considerados durante el análisis del sistema para el desarrollo del software del presente proyecto. Se describirán los métodos utilizados para recopilar, analizar y documentar los requisitos, así como el proceso de análisis y diseño del sistema, incluyendo la creación de diagramas de secuencia, el modelado del dominio y la definición de perfiles de usuario. Finalmente, se identificarán y evaluarán los posibles riesgos que podrían afectar el desarrollo del proyecto, estableciendo estrategias de mitigación para minimizar su impacto.

3.1. Análisis de Requisitos

Los estándares IEEE (Institute of Electrical and Electronics Engineers), son esenciales para garantizar la interoperabilidad, calidad, seguridad y eficiencia en una vasta gama de tecnologías y aplicaciones. Estos estándares, elaborados mediante un proceso de consenso global, abarcan diversas áreas como redes y comunicaciones, ingeniería eléctrica y desarrollo de software. Facilitan la compatibilidad entre productos de diferentes fabricantes, fomentan la innovación y aseguran que los sistemas tecnológicos cumplan con altos niveles de calidad y fiabilidad [34].

Los requisitos son las necesidades del producto que se debe desarrollar para cualquier proyecto de software. Por lo tanto, estas necesidades deben identificarse clara-

mente y registrarse durante la fase de análisis de requisitos. Como resultado de esta etapa, se debe crear un documento de especificación de requisitos que describa las funciones que debe cumplir el futuro sistema [35].

El análisis de requisitos se define como el proceso de investigar las necesidades de los usuarios para determinar los requisitos del sistema, hardware o software, así como el proceso de investigar y refinar dichos requisitos, definición proporcionada por IEEE. Asimismo, el requisito se define como una condición o habilidad que el usuario necesita para resolver un problema o alcanzar un objetivo específico. Esta definición se amplía y se aplica a las condiciones que un sistema o uno de sus componentes debe cumplir para satisfacer un contrato, una norma o una especificación.

Dentro de los requisitos, existen tres tipos de ellos:

- **Requisitos funcionales:** se tratan de declaraciones sobre los servicios que el sistema debe ofrecer, cómo el sistema debe responder a determinadas entradas o cómo el sistema debería comportarse en situaciones particulares. El requisito se plantea desde el punto de vista del sistema. Hay tres perspectivas desde las cuales se deben plasmar los requisitos funcionales, de datos, funcional y de comportamiento.
- **Requisitos no funcionales:** son aspectos del sistema relacionados con la calidad y las restricciones del sistema en su conjunto. Estos requisitos son cruciales para garantizar que el sistema sea eficiente, seguro, escalable y fácil de mantener.
- **Restricciones:** restringen o limitan el proceso de desarrollo o las propiedades del sistema que se está desarrollando. Normalmente, vienen impuestas por los gestores de la organización o por el entorno en el cual el sistema va a funcionar.

3.1.1. Requisitos funcionales

Mapa Principal

1. La aplicación mostrará un mapa interactivo en la página principal con iconos que representan las fincas disponibles.
2. La aplicación permitirá la interacción con los iconos del mapa para ver información básica de cada finca.

Buscador de fincas

1. La aplicación mostrará un buscador de fincas mediante el sistema SIGPAC.
2. La aplicación permitirá filtrar y mostrar resultados de fincas basados en los criterios de búsqueda introducidos.

Información sobre Índices de Vegetación

1. La aplicación mostrará una sección que explique distintos índices de vegetación.
2. La aplicación incluirá un botón que lleve a una vista con información detallada sobre los índices de vegetación.

Tabla de Enfermedades del Olivar

1. La aplicación mostrará una tabla con las enfermedades más comunes del olivar e incluirá un botón que lleve a una vista con información detallada sobre cada enfermedad.

Vista de Fincas

1. La aplicación presentará una lista de fincas con datos detallados.
2. En esta vista, la aplicación permitirá buscar fincas mediante su código SIGPAC.
3. La aplicación incluirá botones de "más información" que lleven a vistas específicas de cada finca.

Detalle de la Finca

1. La aplicación mostrará en el mapa de cada finca específica su contorno y los olivos disponibles.
2. En esta vista, la aplicación mostrará un gráfico sobre las cosechas de los últimos años.
3. La aplicación proporcionará botones para seleccionar años y ver información sobre índices de vegetación, temperaturas medias y lluvias de ese año.

4. La aplicación permitirá la selección de un olivo específico para ver una descripción detallada y los índices de vegetación por año.

3.1.2. Requisitos no funcionales

Rendimiento

1. La aplicación debe cargar el mapa principal y los datos asociados rápido.
2. El buscador de fincas debe devolver resultados correctos y rápido.

Escalabilidad

1. La aplicación debe ser capaz de manejar un aumento en el número de fincas y datos de olivos sin pérdida de rendimiento.

Usabilidad

En nuestro trabajo existen dos roles principalmente, el del administrador de la base de datos, que se encargará de ir alimentándola con datos nuevos y el técnico agrícola, que consulta la página web para saber información sobre su olivar, por tanto, los requisitos de usabilidad serían:

1. La interfaz debe ser intuitiva y fácil de usar, con una navegación clara y coherente.
2. La Base de Datos debe ser clara y fácil de entender.

Confiabilidad y Mantenibilidad

1. Implementar mecanismos de respaldo y recuperación de datos en caso de fallos.
2. El código de la aplicación debe ser modular y bien documentado para facilitar futuras actualizaciones y mantenimiento.
3. Utilizar tecnologías y frameworks ampliamente aceptados y bien soportados.

Compatibilidad

1. La aplicación debe ser compatible con los navegadores web más comunes (Chrome, Firefox, Safari, Edge).

3.1.3. Perfil de usuario

En nuestra aplicación solo se ha definido un perfil de usuario para usar la aplicación, este se corresponde con **Técnicos Agrícolas**.

Estas personas pueden ser dueños de alguna de las parcelas mencionadas previamente o trabajadores con cualificación suficiente como para entender los diferentes factores que se muestran en la página web y cómo pueden afectar a las cosechas. Aunque no es un perfil de usuario, un agricultor avanzado, es decir, con suficiente experiencia, podría también usar la página web efectivamente.

3.2. Análisis y diseño del sistema

El modelado de software es el desarrollo del análisis y diseño de aplicaciones software antes de escribir el código. Esta etapa crucial del desarrollo implica la creación de un conjunto de modelos que representan los cimientos del sistema, incluyendo su estructura y comportamiento interno.

La utilidad del modelado reside en su capacidad para revelar aspectos del sistema que, de otro modo, permanecerían ocultos en el código. Al plasmar las ideas en diagramas y esquemas, se obtiene una visión completa del sistema, facilitando la comprensión de su funcionamiento y la toma de decisiones acertadas.

Al contar con un lenguaje común plasmado en los modelos, se eliminan las ambigüedades y se favorece la colaboración efectiva. Además, estos modelos sirven como documentación que trasciende del proyecto en sí, proporcionando una referencia a largo plazo para futuras modificaciones o mantenimientos.

Destacar que para realizar esta sección se ha usado el programa Visual Paradigm (Fig 3.1). Se trata de una herramienta de modelado de software que proporciona una variedad de características para el diseño y la documentación de sistemas de software. Permite a los desarrolladores crear diagramas UML (Lenguaje de Modelado Unificado), diagramas de flujo de datos, diagramas de casos de uso y más, lo que facilita la

comunicación y la comprensión de los requisitos del sistema.

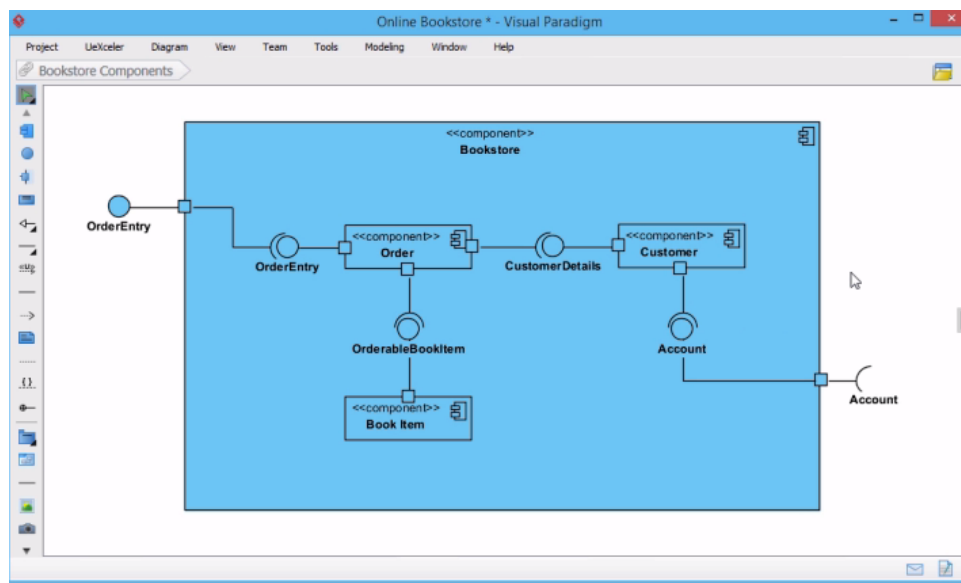


Figura 3.1: Programa Visual Paradigm

3.2.1. Modelo del dominio

El primer paso será realizar el modelo del dominio, este representa las clases conceptuales obtenidas a partir de conceptos o entidades de la parte del mundo real que representa el dominio del problema.

Este permite conocer mejor la aplicación a construir ya que proporciona una visión de la parte del problema que es relevante para la aplicación, visualiza las asociaciones entre conceptos.

En nuestro modelo de dominio intervienen las clases Cosecha, Finca, Zona, Municipio, Provincia, HistóricoFinca, Objeto e HistóricoDatos y se puede ver en la Figura

3.2.2. Casos de Uso

Un Caso de Uso es una lista de pasos, que suelen definir interacciones entre actores y un sistema, para alcanzar un objetivo. Los actores (humanos o sistemas externos) deben ser capaces de tomar decisiones. El sistema queda descrito por un conjunto de casos de uso, cada uno correspondiente a una interacción con el sistema.

Los casos de uso que se han identificado para el técnico agrícola son los siguientes:

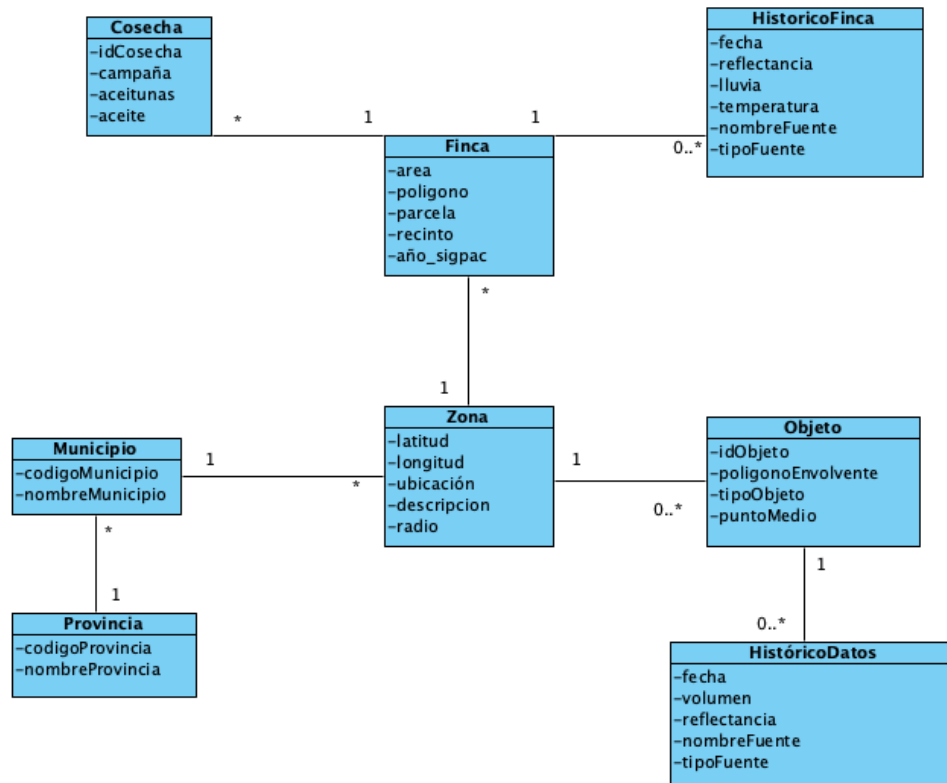


Figura 3.2: Modelo del Dominio - Elaboración Propia

- **Buscar Finca:** proceso que permitirá al usuario buscar parcelas mediante su código SigPac, si el código no es válido o no está almacenado se debe indicar.
- **Ver Datos Parcela:** cuando un código SigPac es válido, el sistema mostrará los datos de la parcela.
- **Ver Datos Cosechas:** cuando se accede a una finca válida, el sistema mostrará los datos almacenados de sus cosechas.
- **Seleccionar Año Histórico:** proceso que permitirá al usuario, cuando la búsqueda de una finca es válida, seleccionar el año del histórico de datos de la finca.
- **Ver Gráficas Histórico:** cuando se seleccione un año, se mostrarán los distintos gráficos sobre el histórico de datos de esa finca.
- **Seleccionar Olivo:** si la finca seleccionada tiene los olivos registrados, esto permitirá al usuario hacer click sobre ellos en el mapa, si no, no dará la opción.
- **Ver Datos Olivo:** si se hace click sobre un olivo válido, se mostrarán todos los datos del olivo en cuestión.

- **Seleccionar Año Histórico Olivo:** si se selecciona el olivo válido, aparecerán distintos años para poder seleccionar y obtener su histórico.
- **Ver Gráficas Histórico Olivo:** si se selecciona un año válido se podrán ver las gráficas del histórico de datos del olivo.
- **Ver Datos Enfermedades:** el usuario podrá acceder a una tabla donde aparezcan las enfermedades más comunes del olivo junto con los posibles factores que pueden hacerlas evidentes y su riesgo.
- **Ver Información Índices de Vegetación:** el usuario podrá ver la información sobre los índices de vegetación que se muestran en las gráficas para que los recuerde y sea consciente del rango de impacto de ellos.

3.2.3. Diagrama de casos de uso

Este diagrama, similar a un mapa que describe el funcionamiento del sistema desde la perspectiva del usuario, representa las interacciones entre el sistema y sus usuarios externos (actores). Cada actor tiene un rol definido y puede realizar acciones específicas dentro del sistema.

Más allá de una simple representación gráfica, el diagrama de casos de uso define las responsabilidades del sistema, es decir, las funcionalidades que debe ofrecer para satisfacer las necesidades de los usuarios. En otras palabras, funciona como un modelo que establece las tareas que el sistema debe cumplir para alcanzar sus objetivos.

Los distintos elementos que podemos encontrar en este diagrama se encuentran explicados a continuación y representados en la Figura 3.3:

- **Actor:** representa a un usuario externo o entidad que interactúa con el sistema. Puede ser una persona, organización o incluso otro sistema o dispositivo. Cada actor tiene un rol definido y un conjunto de acciones que puede realizar dentro del sistema.
- **Sistema:** representa el núcleo del software que se está modelando, es representado por un rectángulo y contiene los casos de uso.
- **Acción / Caso de Uso:** define una función específica que el sistema puede realizar en respuesta a la acción de un actor, se representa como una elipse dentro del sistema.

- **Asociación:** representa la conexión entre un actor y un caso de uso. Se representa como una línea que une al actor con el caso de uso. Indica que el actor puede iniciar o participar en él.
- **Inclusión (Include):** permite modularizar el diagrama, incluyendo un caso de uso dentro de otro. Se representa como una flecha punteada que va desde un caso de uso base a otro incluido.
- **Extensión (Extend):** permite agregar funcionalidad opcional a un caso de uso base. Se representa como una flecha punteada que va desde un actor de un caso de uso base y luego a una extensión que se ejecuta bajo ciertas condiciones específicas.

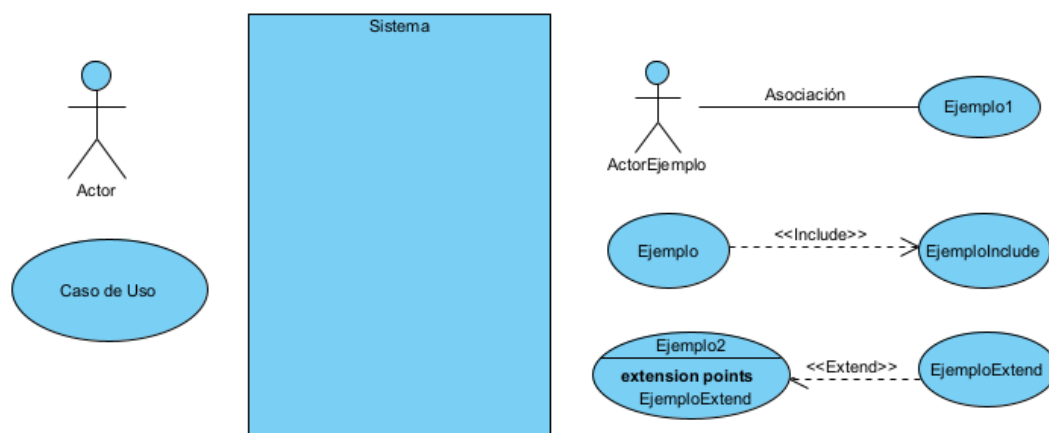


Figura 3.3: Elementos de un Caso de Uso - Elaboración propia

En nuestro proyecto, como se explicó anteriormente, hay un rol de usuario, el técnico agrícola que es el que interactúa con la página web.

Por tanto, su Diagrama de Casos de Uso se puede ver en la Figura 3.4. Este se corresponde a la funcionalidad que tiene el mismo sobre la página web en base a los casos de uso explicados en el apartado anterior.

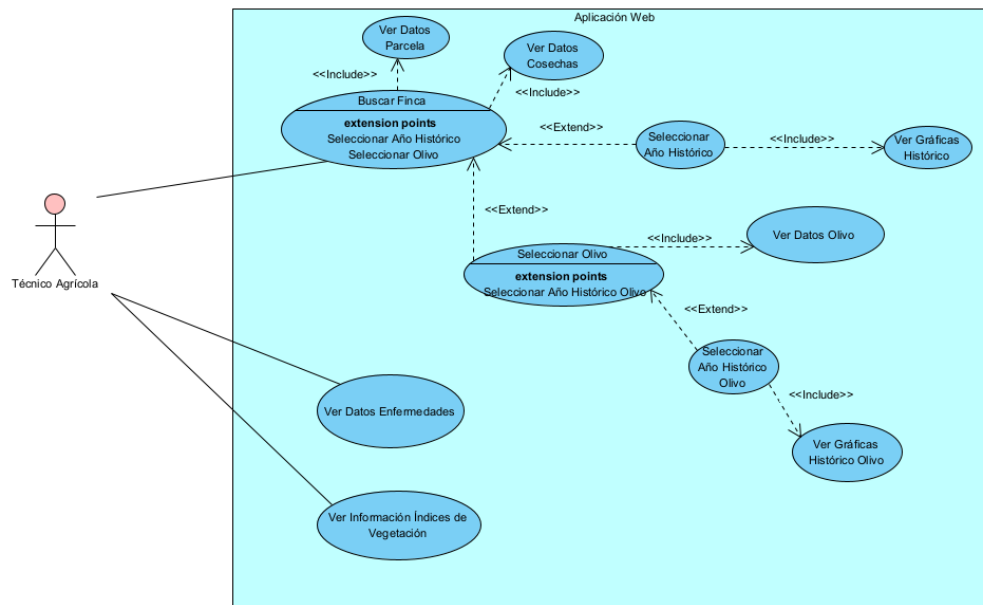


Figura 3.4: Diagrama Casos de Uso - Técnico Agrícola

3.2.4. Diagramas de secuencia del sistema

Los diagramas de secuencia del sistema desarrollarán los escenarios principales de los casos de uso mencionados anteriormente. Estos, son una herramienta de modelado en UML (Lenguaje Unificado de Modelado) que representan la interacción entre objetos en un sistema a lo largo del tiempo. Muestran cómo los objetos se comunican mediante el envío de mensajes en una secuencia temporal específica. Cada diagrama puede contar con diferentes elementos que se explicarán a continuación y, posteriormente, en la Figura 3.5, se podrá ver un ejemplo de ellos.

- **Actor:** entidad externa que interactúa con el sistema. Puede ser un usuario, otro sistema u otra organización.
- **Línea de vida:** representa la existencia de un objeto o un actor a lo largo del tiempo en el diagrama. Se dibuja como una línea vertical discontinua que comienza en un rectángulo (la cabecera) que indica el nombre del objeto o actor.
- **Clase de control:** representa un objeto que controla el flujo de la aplicación. Gestiona la lógica del negocio y coordina la ejecución de tareas entre diferentes objetos del sistema.

- **Clase límite:** representa la interfaz del sistema con los actores. Es responsable de la interacción con el usuario u otros sistemas, como formularios, pantallas, etc.
- **Mensaje:** comunicación entre dos objetos que indica que uno solicita al otro la ejecución de una operación. Se representa con una flecha desde el objeto emisor al receptor.
- **Respuesta:** retorno del mensaje explicado anteriormente, indicando que la operación solicitada ha sido completada. Se suele representar con una flecha discontinua de vuelta desde el receptor al emisor.
- **Mensaje reflexivo:** mensaje que un objeto envía a sí mismo para invocar una de sus propias operaciones. Se representa con una flecha que vuelve al mismo objeto.
- **Operador LOOP:** indica una iteración en el diagrama de secuencia. Encierra los mensajes que deben repetirse en un marco etiquetado como "loopz especifica la condición de repetición.
- **Operador ALT:** representa una elección entre diferentes secuencias de mensajes. Se dibuja como un marco dividido en secciones etiquetadas con condiciones que determinan cuál de las secuencias se ejecutará.
- **Operador OPT:** representa una operación opcional que puede o no ejecutarse, dependiendo de una condición. Se dibuja como un marco etiquetado con .°ptz contiene la secuencia de mensajes que se ejecutará si la condición se cumple.

Como estamos hablando de una Arquitectura Modelo-Vista-Controlador, necesitaremos indicar para cada diagrama la vista y el controlador, para ello se va a usar el **Patrón controlador**, este es un diseño arquitectónico en el que un componente, el controlador, maneja la lógica de la aplicación y actúa como intermediario entre la vista (interfaz de usuario) y el modelo (datos y lógica de negocio). El controlador recibe entradas del usuario, las procesa y actualiza el modelo y la vista en consecuencia. Este patrón promueve una separación clara de responsabilidades, facilitando el mantenimiento y escalabilidad del sistema.

Debido a que tenemos muchos casos de uso que extienden e incluyen otros, para un mayor entendimiento del diagrama completo, se van a explicar los diagramas de secuencia de **Buscar Finca** (incluiría también Ver datos parcela, Ver datos cosechas, Seleccionar año histórico y Ver gráficas histórico), **Seleccionar Olivo** (incluye Ver datos olivo, Seleccionar año histórico olivo y Ver gráficas histórico olivo), **Ver datos enfermedades** y **Ver información índices de vegetación**.

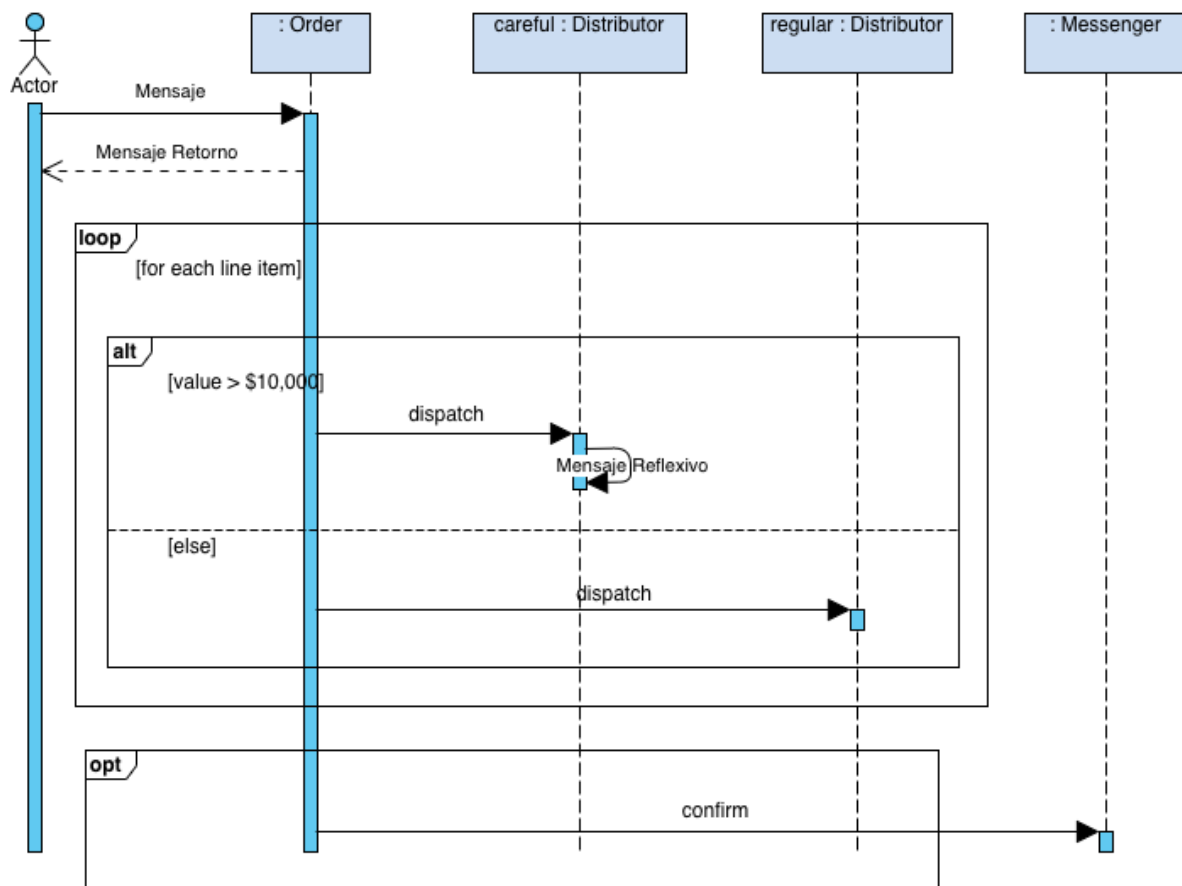


Figura 3.5: Elementos de un Diagrama de Secuencia - Elaboración Propia

Diagrama de Secuencia Buscar Finca

Este es el caso de uso más completo de la aplicación, se ha dividido en dos, este y el siguiente. A continuación se muestra el Diagrama 3.6. Tenemos la Arquitectura Modelo-Vista-Controlador y además se ha aplicado el Patrón controlador en *:Gestor-Fincas*, el administrador se comunica con la vista, esta interactúa con el controlador de la vista y este con el controlador creado.

Diagrama de Secuencia Seleccionar Olivo

Partimos del diagrama de secuencia anterior, si se ha seleccionado una finca válida y esta tiene olivos, se mostrarán. Lo podemos ver en la Figura 3.7.

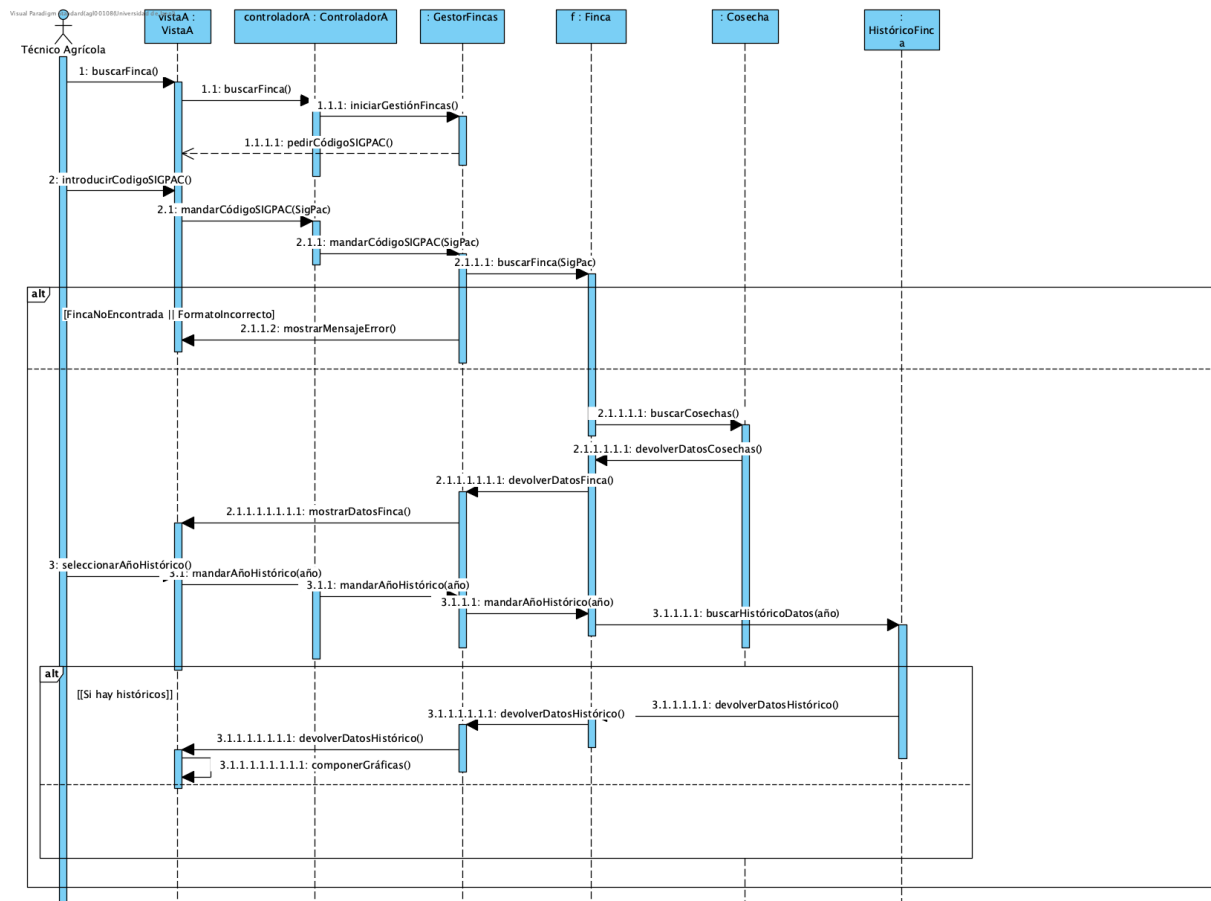


Figura 3.6: Diagrama de Secuencia Buscar Finca - Elaboración Propia

Diagrama de Secuencia Ver Datos Enfermedades

En el siguiente diagrama se muestra el diagrama de secuencia del caso de uso Ver Datos Enfermedades, tenemos la Arquitectura, como en las anteriores, Modelo-Vista-Controlador y además se ha aplicado el Patrón controlador en *:GestiónEnfermedades*, el administrador se comunica con la vista, esta interactúa con el controlador de la vista y este con el controlador creado. Se puede ver en la Figura 3.8.

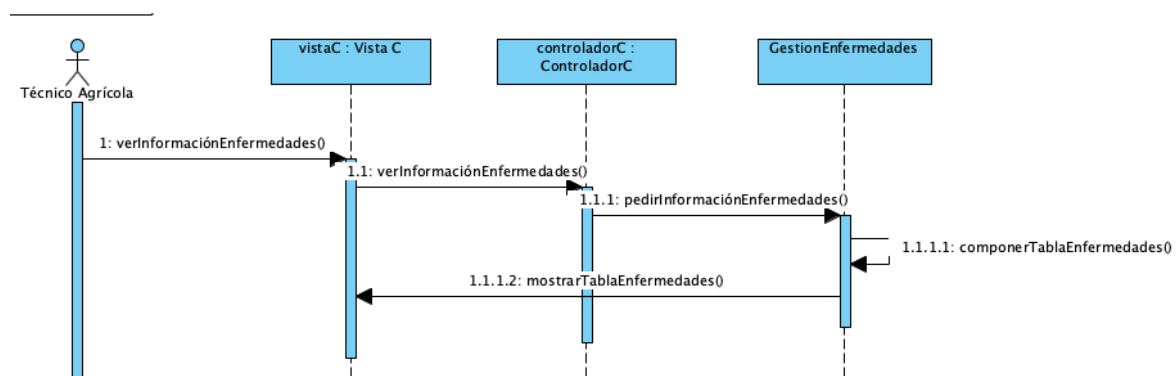


Figura 3.8: Diagrama de Secuencia Ver Datos Enfermedades - Elaboración Propia

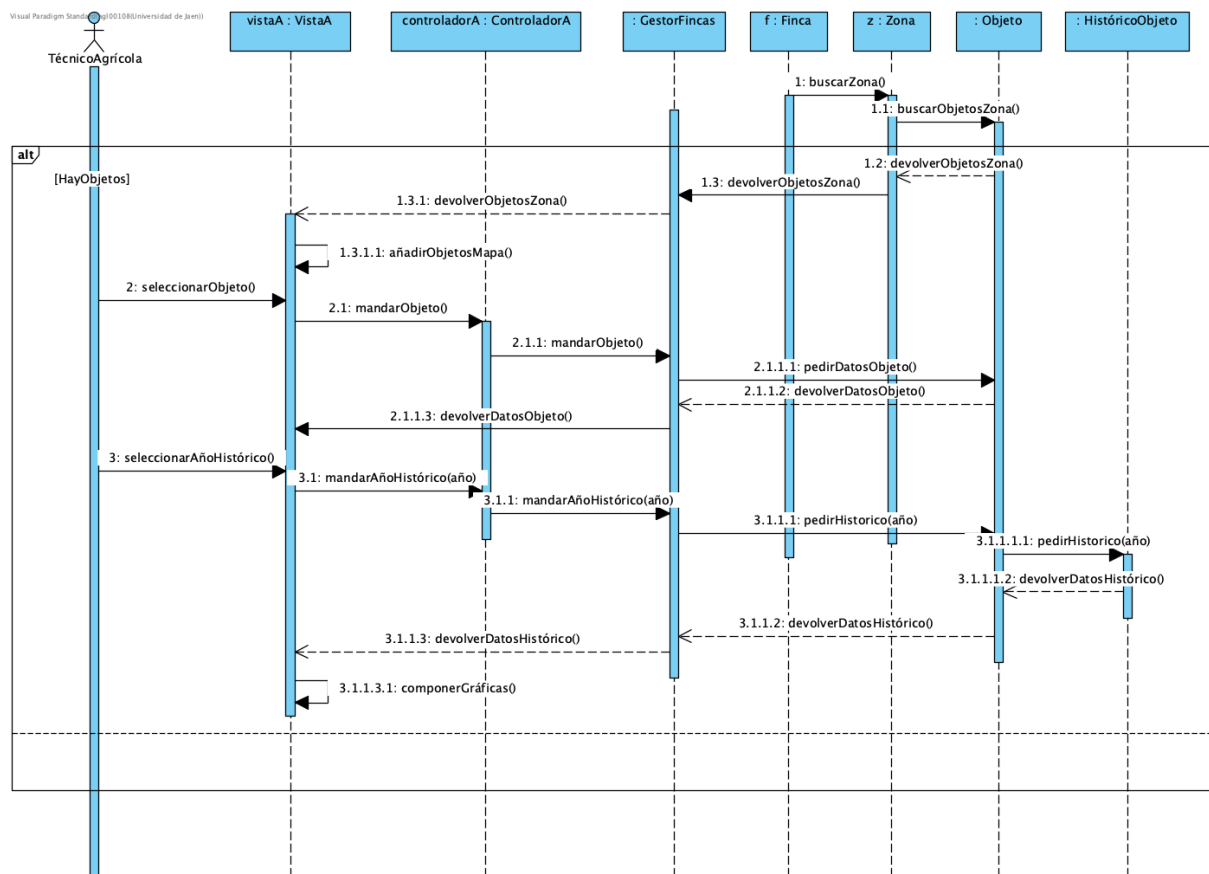


Figura 3.7: Diagrama de Secuencia Buscar Olivo - Elaboración Propia

Diagrama de Secuencia Ver Información Índices de Vegetación

A continuación se mostrará el diagrama de secuencia correspondiente al caso de uso explicado, se puede ver en la Figura 3.9.

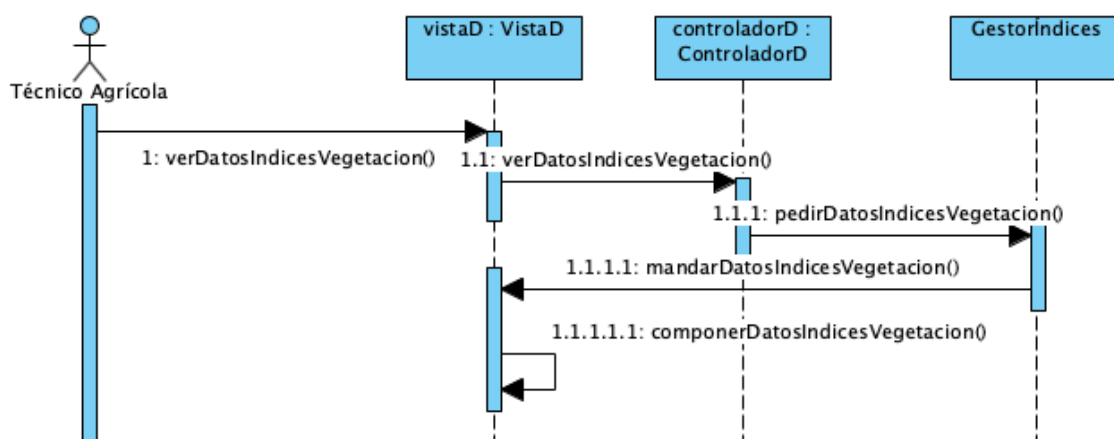


Figura 3.9: Diagrama de Secuencia Ver Información Índices de Vegetación - Elaboración Propia

3.3. Análisis de Riesgos

Como último apartado de esta sección, se va a realizar un análisis de los posibles riesgos que pueden afectar a nuestro proyecto, su alcance, evaluación de impactos y, por último, planes de contingencia. La Tabla 3.10 contiene:

- **Activo:** recurso o componente del sistema que es valioso y necesita protección, como datos, hardware, software, personal, etc.
- **Amenaza:** evento potencial que puede causar daño al activo, como ataques cibernéticos, fallos técnicos, errores humanos, desastres naturales, etc.
- **Probabilidad:** posibilidad de que ocurra una amenaza, evaluada como:
 - **1-Bajo:** la amenaza se materializa a lo sumo una vez al año.
 - **2-Medio:** la amenaza se materializa a lo sumo una vez al mes.
 - **3-Alto:** la amenaza se materializa a lo sumo una vez a la semana.
- **Impacto:** gravedad de las consecuencias si la amenaza se materializa, evaluada como:
 - **1-Bajo:** el daño que deriva la materialización de la amenaza no tiene consecuencias muy relevantes.
 - **2-Medio:** el daño que deriva la materialización de la amenaza tiene consecuencias relevantes.
 - **3-Alto:** el daño que deriva la materialización de la amenaza tiene consecuencias muy relevantes.
- **Riesgo:** combinación de la probabilidad y el impacto de una amenaza, determinando la criticidad del riesgo, evaluado como:
 - **<=4:** el proyecto tiene un riesgo poco reseñable.
 - **>4:** el proyecto tiene riesgo muy reseñable, hay que tratarlo.

Por tanto, observando la Tabla 3.10 podemos ver que los activos a tratar son la Base de Datos, la Documentación y la Página Web.

En cuanto a la Base de Datos, para evitar la pérdida de información, se crearán copias periódicas de ella para que, en caso de ocurrir algún problema, se pueda volver a la situación correcta. Además se harán Commits periódicos para guardar los datos.

ANÁLISIS DE RIESGOS				
Activo	Amenaza	Probabilidad	Impacto	Riesgo
Base de Datos	Pérdida de información	2	3	6
Ordenador	Corte del suministro eléctrico	3	1	3
Documentación	Corrupción de información	1	3	7
Página Web	Fuga de información	2	3	6

Figura 3.10: Tabla Análisis de Riesgos - Elaboración Propia

Por otro lado, la documentación se está haciendo en formato Latex, por ello, se ha usado el programa Overleaf para poder ir realizando el documento en la nube, de forma que se va guardando periódicamente y no tenemos riesgo de perderlo. Aún así, se van descargando de forma local versiones periódicamente.

Por último, tenemos el riesgo de fuga de información de nuestra página web, se ha creado un sistema de versiones de Git, usando GitHub, que garantiza la integridad del proyecto software ya que se encuentra en la nube, permitiendo una vuelta atrás del código si hubiera algún problema.

Una vez hecho el análisis software del sistema, en el siguiente capítulo del proyecto, procederemos a explicar en detalle cada una de las iteraciones mencionadas en la Planificación [2.2](#) y que han contribuido al desarrollo del sistema.

Capítulo 4

Desarrollo del Sistema

4.1. Introducción

En este capítulo se detallarán los pasos seguidos en cada una de las iteraciones para alcanzar el resultado final. Como se comentó en el apartado de planificación, el proceso de desarrollo ha sido iterativo, englobando todas las tareas en cinco iteraciones.

Se han usado muchas tecnologías para poder desarrollar la Aplicación Web, aunque en cada iteración se especificarán las usadas en la Figura 4.1 podemos ver un resumen de ellas:



Figura 4.1: Tecnologías usadas - Elaboración propia

4.1.1. Descripción de la solución propuesta

Esta sección describe brevemente la solución propuesta para resolver el proyecto y cumplir todos los objetivos mencionados en la introducción del proyecto 1.

El desarrollo de la aplicación web se ha dividido en cinco componentes principales: la creación de una Base de Datos implementada en Oracle donde se pudieran representar todas las fincas y sus datos correspondientes, la alimentación de la Base de Datos mencionada a partir de datos obtenidos de satélites y drones, la creación de un backend robusto para obtener toda la información de la Base de Datos y las peticiones REST, la creación de una interfaz de usuario intuitiva y accesible para mostrar toda la información disponible y el análisis de los datos mostrados.

Para garantizar la eficiencia y la confiabilidad de la aplicación, es fundamental implementar un backend robusto. Se ha optado por utilizar Spring Framework para lograrlo. Este Framework es conocido por su capacidad para simplificar el desarrollo de aplicaciones Java al proporcionar una estructura sólida para gestionar dependencias, transacciones y seguridad. Se implementó la capa de negocio, persistencia y un servicio REST para atender todas las peticiones que la interfaz de usuario requiriera.

Por ello, es crucial desarrollar una interfaz coherente y fácil de usar para los usuarios cuyo servicio va destinado. Esta interfaz se sustenta en el backend explicado anteriormente, accediendo a los datos mediante peticiones REST.

El usuario puede explorar la aplicación para ver las parcelas, incluyendo tanto la información geográfica y de uso del suelo como datos sobre sus olivos. La parte visual de la aplicación permite al usuario observar la localización de la finca, su contorno y detalles específicos sobre los olivos.

El usuario, de igual manera, puede ver histórico de datos sobre la parcela, obteniendo información sobre sus cosechas, índices de vegetación, precipitaciones y temperaturas medias. De igual manera, también puede observar información de los olivos individualmente.

Por último, también se han desarrollado vistas de la aplicación de carácter informativo sobre los índices de vegetación que se han usado en la aplicación web al igual que distintas enfermedades relacionadas con el olivo.

4.2. Trabajo inicial

Dentro de este apartado, englobamos todo el procedimiento de decidir qué tecnologías y herramientas se iban a usar para llevar a cabo la aplicación web. De igual modo, se realizó el extenso análisis software mostrado en el Capítulo 3. De esta forma, se tenían todas las herramientas necesarias para empezar a trabajar de una forma más eficaz.

4.3. Primera iteración: Estudio y gestión de datos de entrada

4.3.1. Introducción

En esta primera iteración se empieza a desarrollar la aplicación web, es una de las más extensas puesto que se necesitan mucha información para alimentar la Base de Datos, desarrollada en siguiente Iteración 4.4. Como se mencionó anteriormente, ambas se realizaron paralelamente ya que necesitaban datos una de la otra.

El código necesario para esta iteración, se puede encontrar en el siguiente repositorio de [GitHub](#). Ya que se ha creado un programa en Java con diferentes clases que realizarán la conversión, podemos verlo en la Figura 4.2, no incluye relaciones puesto que cada clase tiene su trabajo específico, como se explicará durante la iteración.

Estudios de alternativas y viabilidad

En esta iteración, en primer lugar, se estudiaron los diferentes índices de vegetación. Finalmente, se eligieron, debido a su capacidad para monitorear la salud y el vigor de las plantas, los índices NDVI, NDMI y SAVI, estos se explicaron anteriormente en el Apartado 1.7.2.

El primero mide la salud general de la vegetación y su densidad, el segundo es similar al NDVI, pero ajusta los efectos del suelo, lo que es útil en áreas con escasa cobertura vegetal y el NDMI evalúa el contenido de agua en la vegetación, importante para la gestión del riego. Juntos, estos índices proporcionan una visión completa del estado de los olivos, ayudando a optimizar la producción y la gestión del cultivo.

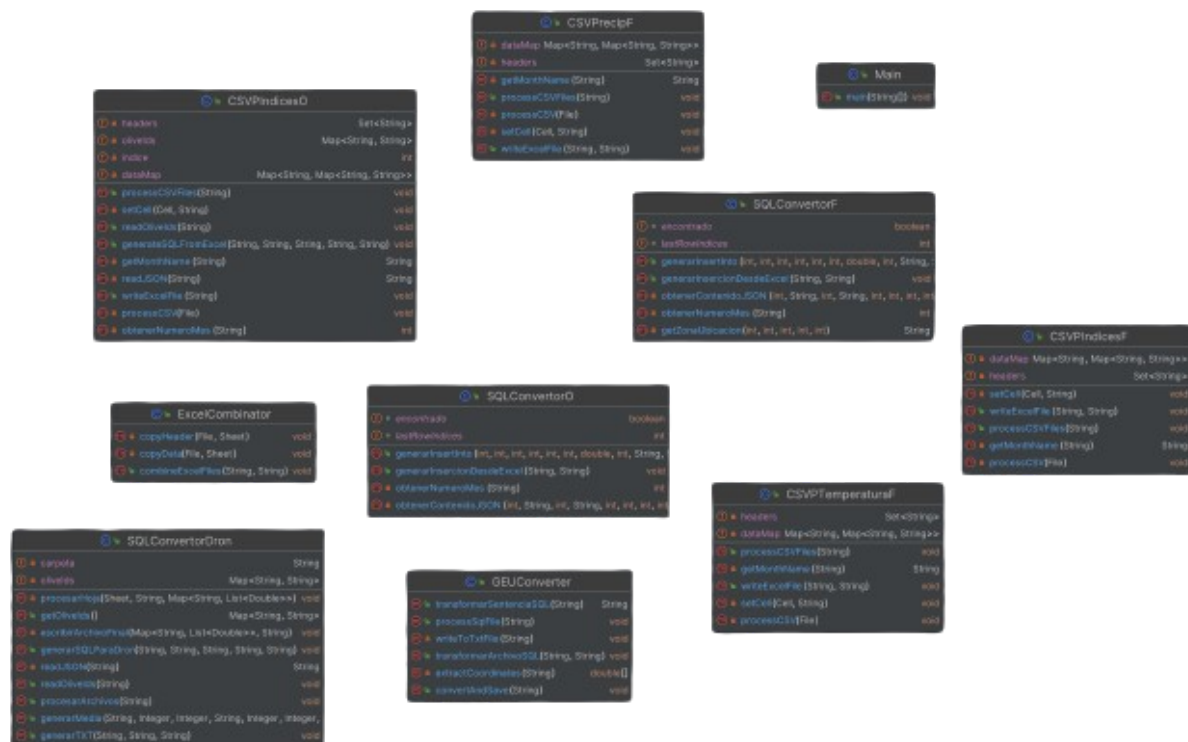


Figura 4.2: Diagrama UML Transformador - Elaboración propia

En cuanto a los demás factores, se observaron los que afectaban en mayor parte a las enfermedades estudiadas en el Apartado 1.7.4. Estos eran las temperaturas y la humedad, que se puede obtener a partir de las precipitaciones.

Tecnologías Usadas

A continuación se explicarán todas las tecnologías usadas:

■ Lenguajes de programación

- **Java:** lenguaje de programación de propósito general, orientado a objetos y con una fuerte tipificación. Es ampliamente utilizado en el desarrollo de aplicaciones empresariales, aplicaciones móviles y sistemas de gran escala debido a su robustez, seguridad y portabilidad. Se ha usado mucho en la adaptación de los datos de distintas fuentes para poder conseguir un resultado aceptado en la Base de Datos. Además, debido a que Spring es un Framework desarrollado en este lenguaje, se ha usado para crear todo el backend de la aplicación.
- **Python:** lenguaje de programación de alto nivel, conocido por su legibilidad y simplicidad. Es muy utilizado en diversos campos como desarrollo web,

automatización, análisis de datos, inteligencia artificial y aprendizaje automático debido a su extensa biblioteca estándar y la comunidad activa de desarrolladores. Se ha usado en la obtención de datos con Google Earth Engine ya que es el lenguaje de programación principal.

■ Bibliotecas Java

- **Apache POI:** biblioteca de Java que proporciona clases y métodos para trabajar con documentos de Microsoft Office, como Excel. Se ha usado `Workbook`, `XSSFWorkbook` para crear un nuevo libro de Excel, `Sheet` para crear una nueva hoja dentro del libro, `Row` y `Cell` para agregar filas y celdas en la hoja de Excel y `FileOutputStream` para escribir el contenido del libro de Excel en un archivo.
- **OpenCSV:** biblioteca de Java que proporciona funcionalidades para leer y escribir archivos CSV en Java. Se utilizó `CSVReader` para leer los datos de los archivos CSV y `readAll()` para leer todos los datos del CSV y almacenarlos en una lista.
- **Java IO:** proporciona clases para la entrada y salida de datos en Java. Incluye clases para trabajar con archivos, flujos de entrada y salida, y clases para la lectura y escritura de datos de manera eficiente. Se usó `File`, `FileInputStream`, `FileOutputStream`, `BufferedReader`, `BufferedWriter`, `FileReader`, `FileWriter` para leer y escribir archivos.
- **Java Util:** proporciona clases e interfaces para estructuras de datos y utilidades comunes en Java. Se ha usado `Map`, `HashMap`, `List`, `ArrayList` para almacenar y manipular colecciones de datos.
- **JSON:** proporciona clases y métodos para leer y escribir datos en formato JSON. Se ha usado `JSONObject`, `JSONArray` para manejar datos en formato JSON.

■ Entornos de desarrollo

- **IntelliJ (Fig 4.3):** dado que el software se desarrollaría en Java, el entorno de desarrollo integrado (IDE) principal de la aplicación fue IntelliJ de JetBrains. La inteligencia artificial y las capacidades de análisis estático de IntelliJ son excelentes porque ayudan a los desarrolladores a escribir código de manera más rápida y eficiente. Sugerencias de código, refactorización automática, detección de errores en tiempo real y muchas otras funciones mejoran la productividad de los desarrolladores.

Además, IntelliJ proporciona una amplia variedad de extensiones y complementos que permiten a los desarrolladores adaptar el IDE a sus necesida-

des particulares. Estos complementos pueden ayudar con marcos y tecnologías específicos, facilitar la integración con herramientas de desarrollo de terceros o agregar características adicionales a las herramientas de desarrollo.

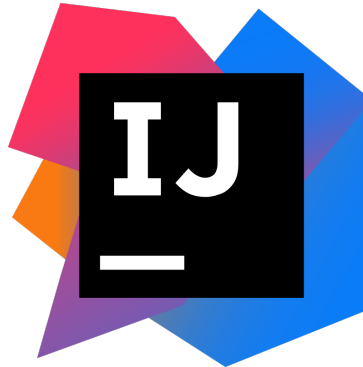


Figura 4.3: IntelliJ Logo

- **Google Earth Engine (Fig 4.4):** para obtener todos los datos necesarios de índices de vegetación, temperatura y precipitaciones se ha programado en Google Earth Engine, es una plataforma en la nube para el análisis y visualización de datos geospaciales. Desarrollada por Google, ofrece acceso a una enorme cantidad de datos de imágenes satelitales y otras fuentes geoespaciales, así como herramientas de análisis y visualización avanzadas. Para realizar cálculos distribuidos en paralelo, utiliza la infraestructura de Google Cloud Platform, lo que permite realizar análisis complejos en cuestión de segundos o minutos, en lugar de horas o días como sería necesario utilizando métodos tradicionales. Se usa, sobre todo, Python.



Figura 4.4: Google Earth Engine - Gis Beers

■ Programas de terceros

- **QGIS (Fig 4.5):** es un SIG de código abierto que ofrece herramientas avanzadas para visualizar, analizar y editar datos geoespaciales. En campos

como la cartografía, la planificación urbana, la gestión de recursos naturales y la investigación científica, es ampliamente utilizado. Se ha usado en el proyecto para obtener todos los datos del histórico tanto de fincas como de olivos procedentes de dron. Además, fue indispensable a la hora de hacer la segmentación de fincas en olivos.

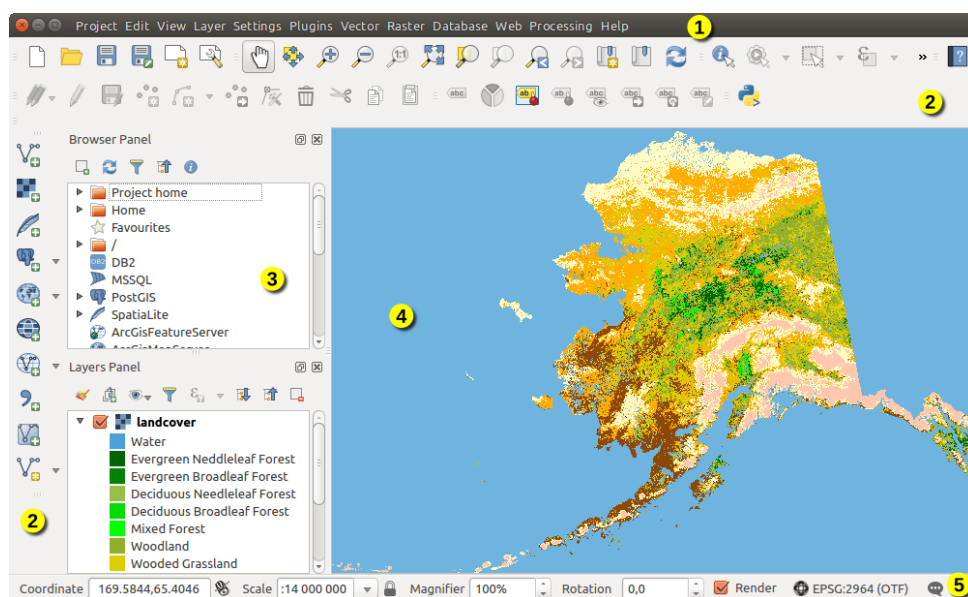


Figura 4.5: QGis

- **Pentaho Spoon (Fig 4.6):** también conocido como Pentaho Data Integration (PDI), es una herramienta de integración de datos de código abierto que permite la extracción, transformación y carga (ETL) de datos desde una variedad de fuentes a una variedad de lugares. Es un programa de Pentaho Business Analytics que se utiliza para realizar procesos de integración de datos complejos en entornos comerciales. En el proyecto, se usó para pasar de formato excel a JSON, generando claves y datos que luego se introdujeron en la Base de Datos en las tablas HISTORICO_DATOS e HISTORICO_FINCA.

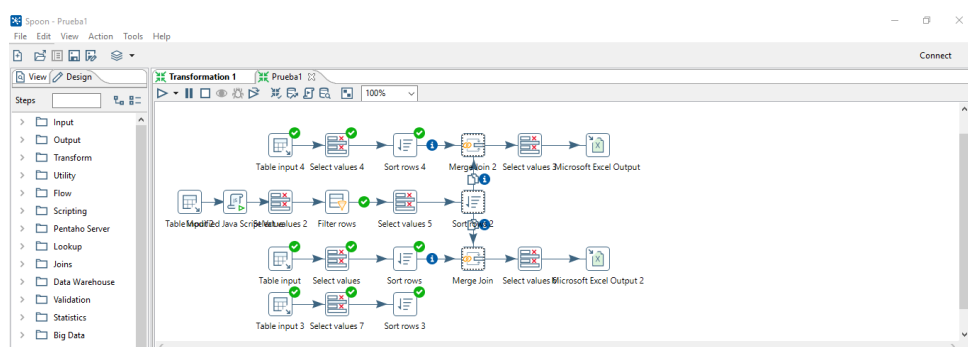


Figura 4.6: Pentaho Spoon

4.3.2. Obtención de Fincas y Olivos

El primer paso fue obtener las fincas a estudiar. Estas, debían estar clasificadas mediante su código SIGPAC, como se explicó en el Apartado 1.5. La obtención de estas se produjo desde dos fuentes distintas:

1. **Fincas J1, J2, J3 y J4:** estas fincas y todos sus datos correspondientes fueron suministrados por el grupo de investigación ya que se corresponden a parcelas que se están estudiando para otros fines. Los datos de estas fincas, de las que se obtiene mayor información, son los siguientes:
 - **J1:** se trata de una finca situada en Marmolejo, con 1.0558 hectáreas. Su código SIGPAC es 1:23:59:0:0:6:211:1.
 - **J2:** se trata de una finca situada Andújar, con un área de 4.289 hectáreas. Su código SIGPAC es 1:23:0:0:7:325:2.
 - **J3:** es otra finca situada en Andújar, cuya área es de 5.1978 hectáreas y su código SIGPAC es 1:23:0:0:8:101:2.
 - **J4:** esta finca está situada en Espeluy, tiene 12.3165 hectáreas de extensión y su código SIGPAC es 1:23:0:0:3:101:3.
2. Con el fin de obtener la mayor información posible y de poder hacer pruebas con un usuario real, se agregaron 47 fincas más, correspondientes a los municipios de Torredonjimeno e Higuera de Calatrava. Para una mayor claridad, también se nos proporcionó las distintas zonas en las que se ubicaban las fincas con su nombre común, para poder clasificarlas, se nombraron como Tx donde x es el número, a todas las zonas procedentes de Torredonjimeno y Hx las de Higuera de Calatrava.

Finalmente, en nuestra base de datos, se contó con 51 fincas diferentes de las que poder extraer información. Entre todas ellas, se seleccionaron J1, J2 Y J3 para realizar el proceso de segmentación, que se explicará a continuación, y poder obtener sus olivos.

Segmentación de Olivos con GEU

El software **GEU (Geospatial and Environmental tools of University of Jaén)** representa un avance significativo en la aplicación de tecnología para el estudio y análisis de entornos naturales, específicamente en la segmentación de plantas en

áreas como olivares, viñedos y bosques. Desarrollado como resultado de una tesis doctoral defendida en la Universidad de Jaén por Juan Manuel Jurado Rodríguez, GEU ofrece una plataforma integral que combina herramientas avanzadas de análisis de imágenes con técnicas de modelado 3D.

GEU permite la digitalización de entornos naturales mediante la captación de imágenes de alta resolución tomadas por drones equipados con cámaras espectrales, térmicas y RGB. La integración de esta información en un contexto tridimensional brinda a los usuarios la capacidad de visualizar y analizar detalladamente el área de estudio.

Una de las características destacadas de GEU en nuestro trabajo, es su capacidad para segmentar y caracterizar automáticamente las plantas en áreas como olivares. Esto se logra mediante el análisis de variables morfológicas y espectrales significativas, lo que proporciona valiosa información para la gestión agrícola, como la predicción de cosechas, el monitoreo de la salud de los cultivos y la detección temprana de plagas.

Para poder realizar de manera correcta la segmentación, necesitamos que la imagen de la finca sea de alto contraste, bien generada desde una nube de puntos o umbralizada mediante la herramienta QGis que se explicará más adelante. Como ejemplo, se tomará la Imagen [4.7](#).

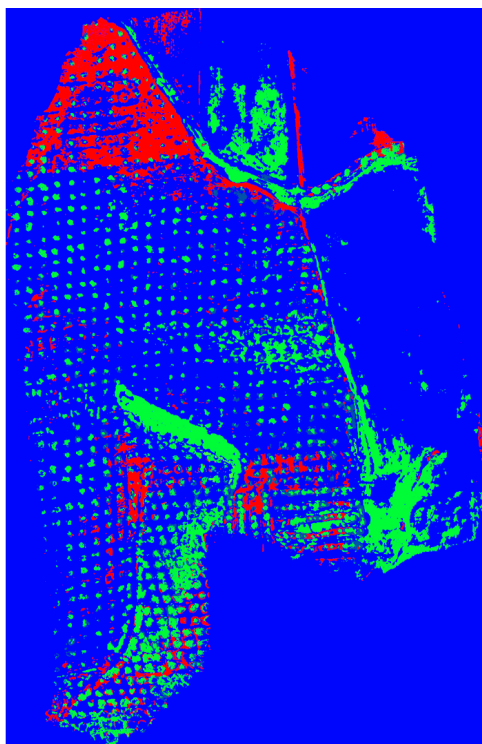


Figura 4.7: Imagen de alto contraste J2 - Grupo de Investigación

Esta imagen proviene de la nube de puntos obtenida mediante el proceso de **Struc-**

ture from Motion (SfM) a partir de capturas aéreas realizadas con drones. Esta nube de puntos ha sido previamente clasificada para asignar a cada punto una categoría correspondiente, como suelo, vegetación o edificaciones. Posteriormente, esta nube clasificada se ha proyectado sobre un plano, resultando en imágenes bidimensionales que se utilizan como entrada para el algoritmo de segmentación.

Debemos seleccionar el canal de color, en este caso sería la G, verde, ya que si observamos la imagen nuestros olivos están destacados en este color. En “parámetros de vegetación” debemos seleccionar el Umbral, que nos indica a partir de qué tono del color seleccionado se debe proceder a la segmentación. La “búsqueda de contornos” sirve para la dilatación de la imagen, en condiciones normales, no haría falta alterar lo predeterminado, vemos cómo se ve en la Figura 4.8.

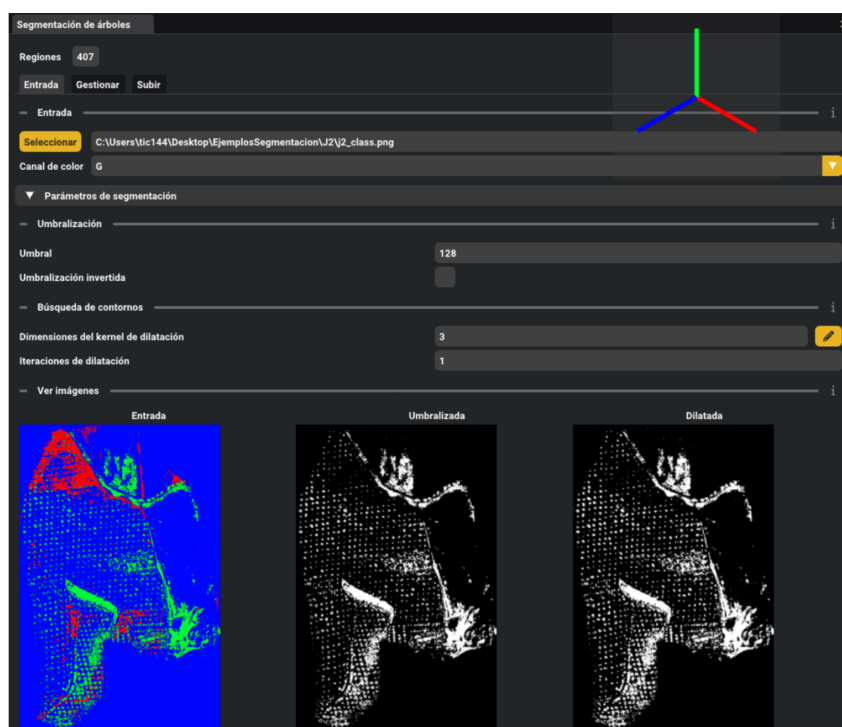


Figura 4.8: GEU Ejemplo - Elaboración Propia

A continuación, debemos movernos al apartado “Gestionar”, en “Tamaño de segmentos” debemos modificar los parámetros de la caja pequeña y grande de tal forma que nuestros olivos estén entre esos umbrales. Procedemos a segmentar y nos hará una primera iteración. Si no estamos convencidos, debemos alterar los campos anteriormente dichos.

La herramienta tiene una última opción en la que podemos añadir, eliminar o mover algunas cajas que se han generado para nuestros árboles.

Por último, en la opción “Subir”, debemos indicar los parámetros de nuestra fin-

ca (esquina superior derecha e inferior izquierda, nombre y ubicación) y tenemos la opción de subirlo a la Base de Datos del grupo o descargarnos el .sql con todas las coordenadas. Para nuestro ejemplo, seleccionamos la segunda opción.

Una vez se haya perfilado la tabla Objeto en la segunda iteración, debemos tener en cuenta que GEU genera sentencias SQL adaptadas a la Base de Datos del grupo de investigación, por ello, hay que hacer unos pequeños cambios, añadiendo el punto medio de los objetos, el código de la provincia y el municipio.

Para ello, se creó una clase en Java llamada **GEUConverter** que, recibiendo los datos generados anteriormente, nos genere, por un lado, un documento en formato SQL con todos los INSERT INTO adaptados y, por otro lado, un documento con todos los puntos medios de la caja envolvente de cada Olivo que nos será útil en posteriores apartados. A continuación se describe los métodos de la clase **GEUConverter**, que se puede encontrar en el repositorio del trabajo.

- **processSqlFile(String sqlFilePath):** esta función lee un archivo SQL línea por línea y llama a *convertAndSave(String insertStatement)* si encuentra una línea que comienza con INSERT INTO OBJETO.
- **convertAndSave(String insertStatement):** esta función toma una sentencia INSERT INTO OBJETO y extrae las coordenadas del POLIGONO_ENVOLVENTE. Luego, calcula el punto medio (centro) de estas coordenadas y lo guarda en un archivo de texto.
- **writeToTxtFile(String centralPoint):** esta función es utilizada por *convertAndSave* para escribir el punto medio en un archivo de texto.
- **transformarArchivoSQL(String inputFile, String outputFile):** esta función toma un archivo de entrada que contiene sentencias SQL, transforma cada sentencia INSERT INTO OBJETO utilizando *transformarSentenciaSQL(String sqlStatement)*, y escribe las sentencias transformadas en un archivo de salida.
- **transformarSentenciaSQL(String sqlStatement):** esta función transforma una sentencia INSERT INTO OBJETO al formato deseado. Extrae información como el tipo de objeto, la ubicación, el código del municipio, el polígono envolvente, y luego calcula el punto medio. Luego, genera una nueva sentencia INSERT INTO OBJETO con el punto medio incluido.
- **extractCoordinates(String insertStatement):** esta función extrae las coordenadas del polígono envolvente de una sentencia INSERT INTO OBJETO. Actual-

mente, asume que las coordenadas están en un formato específico y las extrae utilizando una técnica simple de división de cadenas.

Umbralización de imágenes con índice SAVI

Hay casos en los que las nubes de puntos son muy grandes para poder generar una imagen de alto contraste. A continuación propongo una serie de pasos alternativos para poder generar una imagen lo más clara posible para poder introducirla en GEU y realizar la segmentación.

Necesitamos una imagen tomada por un dron de la finca representando el índice de vegetación SAVI. Se ha elegido este ya que es especialmente adecuado para estudios que analizan la vegetación en etapas tempranas de crecimiento o en áreas con vegetación dispersa. En general, el SAVI puede considerarse como una alternativa viable cuando se encuentran suelos con una baja densidad vegetal y la exposición de la superficie del suelo es relevante. Es importante recalcar que si las imágenes se corresponden a estaciones como verano, donde no hay ruido de otra vegetación en el suelo, los resultados serán mejores.

A continuación voy a describir el proceso para pasar a una imagen como la Figura 4.9, donde se aprecia en color verde la vegetación y en color azul la parte sin ella.

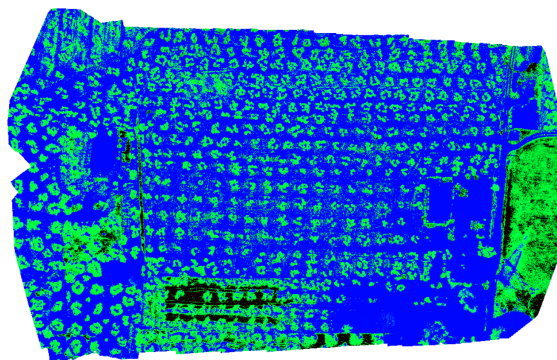


Figura 4.9: Umbralización Índice SAVI - Elaboración propia

Como se ha dicho anteriormente, necesitamos una imagen georeferenciada en formato TIF representando el índice de vegetación SAVI, la incluimos como capa nueva en el programa QGis, debemos pulsar en la capa y en el menú, seleccionar Pseudocolor Monobanda en Tipo de Renderizador, la interpolación será lineal, el modo intervalo igual y ponemos 10 clases.

Ahora, se debe ir observando el rango del índice que nos interesa, ya que lo que el programa está haciendo es asignar un color al píxel según el índice de vegetación

que tenga asignado. Así pues, si vemos que el rango de vegetación del olivo está, por ejemplo, entre los valores 0.1 y 0.3, ponemos ese rango a verde más fuerte, descartando otras zonas del suelo azul y la zona verde más concentrada, se puede tomar la opción de ponerla negra pensando en que el programa no lo detectase.

Una vez terminado el proceso, debemos descargar la imagen en formato PNG y realizar el procedimiento de segmentación en GEU explicado anteriormente.

Obtención de Shapefile para Olivos

Previamente se explicó que necesitábamos los puntos medios de los olivos, esta será nuestra entrada para poder conseguir el archivo Shapefile de los olivos para poder obtener los datos satelitales en Google Earth Engine. El programa para hacerlo será QGIS.

En primer lugar, cargamos en una capa la imagen georeferenciada de la parcela que represente el índice de vegetación SAVI, esta etapa no es obligatoria pero sí nos ayudará a ver si los puntos que generemos corresponden con el olivar. Necesitamos, de igual modo, los límites de la finca en formato ESRI Shapefile (SHP), descargados de la página del SigPac en apartados anteriores. El resultado se corresponde a la Figura 4.10.

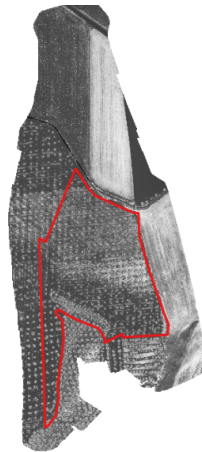


Figura 4.10: Ejemplo de contorno de finca en QGIS - Elaboración Propia

Ahora, es momento de cargar los puntos medios de los olivos. Accedemos a Añadir capa de texto delimitado en el menú Capa, Añadir capa. En definición de geometría, seleccionamos el campo X y el campo Y.

Al mostrarse los valores, podemos observar que aparecen los puntos medios de

los olivos pero no representan el árbol en su totalidad, además, puede haber algunos valores que se encuentren fuera de los límites. Por tanto, vamos a hacer un recorte de ellos y vamos a dejar los que estén en el interior y posteriormente vamos a hacer una dilatación.

Para hacer el recorte de los puntos, accedemos a Vectorial, Herramientas de geoproceso y Cortar. Nos quedaría una nueva capa con los olivos dentro de los límites del recinto como vemos en la Figura 4.11

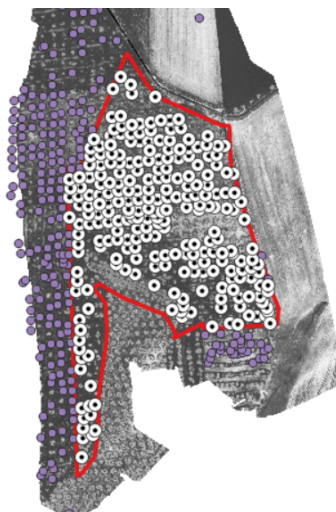


Figura 4.11: Ejemplo de recorte de olivos en QGIS - Elaboración Propia

Una vez cortados, debemos dilatar los puntos. Para ello, pulsamos “Vectorial”, “Herramientas de geoproceso” y “Buffer”. Elegimos la capa de los puntos cortados, en distancia ponemos la cantidad de dilatación de los puntos, yo he elegido 2 metros pues creo que es suficiente para representar el olivo. Se ve el resultado en la Figura 4.12.

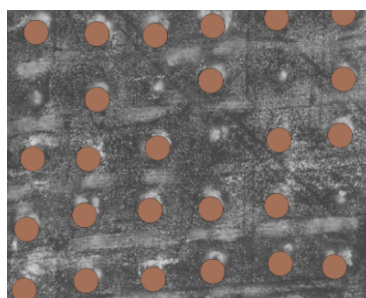


Figura 4.12: Ejemplo de dilatación de olivos en QGIS - Elaboración Propia

Por último, tenemos que exportar la capa de los puntos que acabamos de generar, para ello debemos pulsar el botón derecho sobre la capa, “Exportar”, “Guardar objetos como” y seleccionar “Archivo shape de ESRI”. Ya tendríamos todo listo para poder obtener la información de los olivos.

4.3.3. Obtención de Factores

Tras obtener todos los datos correspondientes a las fincas y sus olivos, debemos seleccionar los factores a estudiar. Se ha explicado anteriormente el análisis de las distintas enfermedades que puede padecer el olivar y las más probables. Observando la Tabla 1.19 se llegó a la conclusión, viendo los factores que hacían evidente su aparición, de que debíamos estudiar los índices de vegetación SAVI, NDMI Y NDVI, además, sería de vital importancia las precipitaciones que hubieran caído en las distintas fincas y las temperaturas máximas, medias y mínimas tanto para la noche como el día.

Los datos pueden provenir de dos fuentes, satelital, usando Google Earth Engine o por imágenes georeferenciadas normalizadas de los índices de vegetación obtenidas por drones del grupo de investigación, a continuación se describen los dos procesos.

Obtención de datos en Google Earth Engine

Para poder trabajar con Google Earth Engine, en primer lugar, necesitamos el contorno de las fincas cuyos datos queremos obtener, esto se puede conseguir en el [Visor SigPac del Gobierno de España](#). Para ello, hacemos una búsqueda progresiva de cada una de las fincas mediante su código SigPac, seleccionamos en el menú de capas sólo la opción Recinto, pulsamos la opción Geometría en descargas y vamos añadiendo cada una de las parcelas, eligiendo el formato Shapefile. Un posible resultado se puede ver en la Figura 4.13.

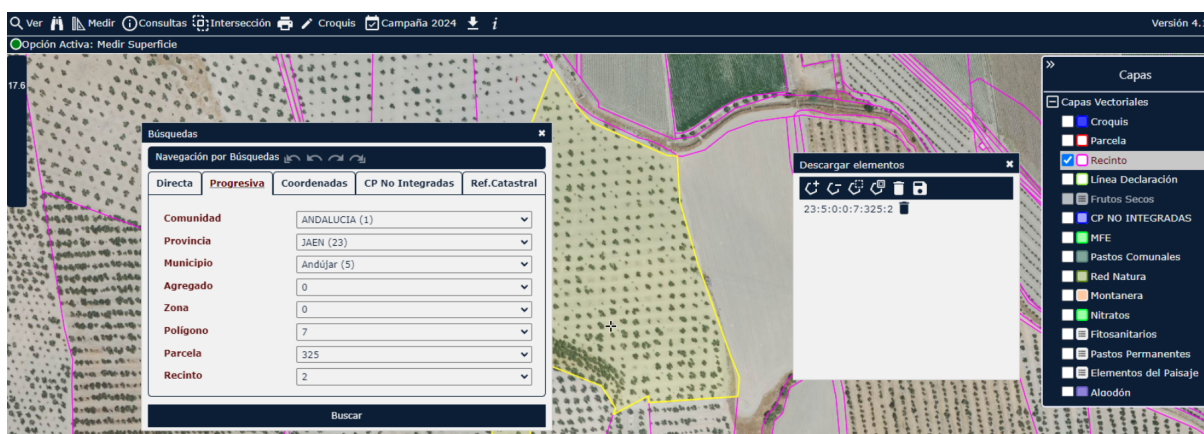


Figura 4.13: Ejemplo descarga de geometría SigPac - Elaboración Propia

Una vez descargados los datos, entramos a la aplicación de Google Earth Engine. Para incluir los datos de nuestras fincas, pulsamos sobre Assets, New, Shape files y

seleccionamos nuestro archivo fuente. Hay que incluir todos los archivos de la carpeta descargada. Damos a upload y nos aparecerán como se ve en la Figura 4.14. Ya tendríamos todo listo para empezar a programar.

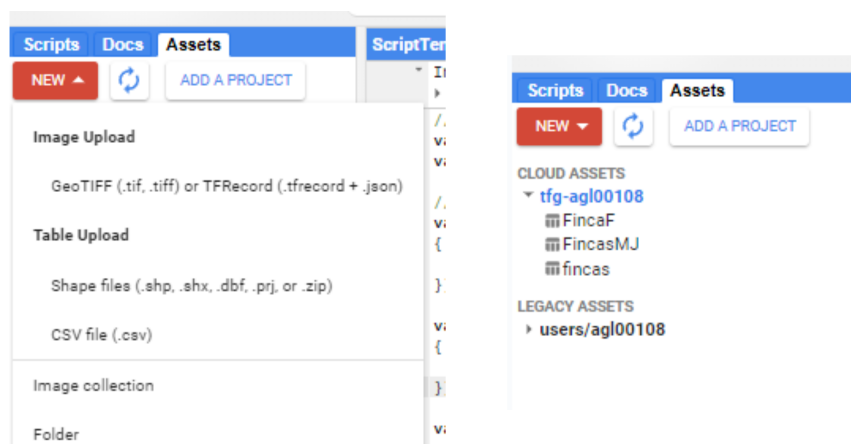


Figura 4.14: Carga de datos en Google Earth Engine - Elaboración Propia

Obtención Índices de Vegetación

El entorno de programación de Google Earth Engine está basado en Python, por tanto, todo el código que se mostrará a continuación está en este lenguaje de programación. El satélite usado para obtener los datos es Sentinel-2, cuyas características se explicaron en la introducción.

También cabe destacar que este algoritmo es válido tanto para las fincas como sus correspondientes olivos, solo cambiaría la zona a declarar al principio del código.

En primer lugar, debemos definir las fechas, es decir el año del que queremos obtener los datos junto con las fechas de inicio y fin (Listado 4.1). Definimos la colección de Sentinel 2 nivel 2 para filtrar imágenes en el período de enero en nuestra región específica con menos del 20 % de píxeles nublados ya que, si no, nuestros cálculos no serían lo suficientemente precisos.

```

1 var ColeccionSentinel = ee.ImageCollection("COPERNICUS/S2")
2   .filterDate(startdate, enddate)
3   .filterBounds(zona)
4   .filterMetadata('CLOUDY_PIXEL_PERCENTAGE', 'less_than', 20);

```

Listado 4.1: Definición de colección de Sentinel-2

Posteriormente (Listado 4.2), obtenemos la media de imágenes de enero y la cantidad de ellas para evaluación de posibles errores si no existieran.

```

1 var Vegetacion = ee.Image(ColeccionSentinel.mean()).clip(zona);
2 print('Media_imágenes_enero', Vegetacion);
3 var cantidad = ColeccionSentinel.size();
4 print('Cantidad_imágenes_enero', cantidad);

```

Listado 4.2: Obtención de número de imágenes

En la parte que se presenta a continuación (Listado 4.3), se realiza los cálculos de los índices de vegetación seleccionados previamente utilizando las bandas espectrales de las imágenes. En cuanto a las bandas, B8 se corresponde con NIR, B4 con la banda RED y B11 con SWIR.

```

1 // NDVI (Normalized Difference Vegetation Index)
2 var NDVI = Vegetacion.expression('float_((NIR_ - RED_) / (NIR_ + RED_))',
3 {
4   'NIR': Vegetacion.select('B8'),
5   'RED': Vegetacion.select('B4')});
6
7 // SAVI (Soil Adjusted Vegetation Index)
8 var SAVI = Vegetacion1.expression ('float_(((NIR_ - RED_) / (NIR_ + RED_ + L)) * (1 + L))',
9 {
10   'L': 0.5, // Cobertura vegetacion 0-1
11   'NIR': Vegetacion.select ('B8'),
12   'RED': Vegetacion.select ('B4')});
13 // NDMI (Normalized Difference Moisture Index)
14 var NDMI = Vegetacion1.expression('float_((NIR_ - SWIR_) / (NIR_ + SWIR_))',
15 {
16   'NIR': Vegetacion.select('B8'),
17   'SWIR': Vegetacion.select('B11')});

```

Listado 4.3: Cálculo de índices de vegetación

Aquí (Listado 4.4), añadimos las capas al mapa, incluyendo una capa de imágenes RGB naturales y capas para visualizar NDVI, SAVI y NDMI. Además, se reducen las regiones de las imágenes para obtener estadísticas (media, percentiles) en la región especificada (zona).

```

1 // Representacion de indices de vegetacion
2 Map.addLayer (Vegetacion, {max: 4000.0, min: 0.0, gamma: 1.0, bands: ['B4', 'B3', 'B2']}, '
   RGB_color_natural', 0);
3 Map.addLayer (NDVI, color, 'NDVI_ene', 1);
4 Map.addLayer (SAVI Simbologia, 'SAVI_ene', 0);
5 Map.addLayer (NDMI, colorWater, 'NDMI_ene', 0);
6
7 var reducers = ee.Reducer.mean().combine({
8   reducer2: ee.Reducer.percentile([25, 50, 75]),
9   sharedInputs: true,
10 });

```

Listado 4.4: Añadir capas de índices

Por último (Listado 4.5), depuramos las propiedades de las features para exportar

los resultados en forma de tablas CSV para NDVI, SAVI y DNMI. También se podrían descargar las imágenes en formato GeoTIFF pero para este proyecto no va a hacer falta.

```
1 var MeansOfFeatures1 = NDVI.reduceRegions
2 ({
3   collection: zona,
4   reducer: reducers,
5   scale: 20,
6 });
7 var MeansOfFeatures2 = SAVI.reduceRegions
8 ({
9   collection: zona,
10  reducer: reducers,
11  scale: 20,
12 });
13 var MeansOfFeatures3 = NDVI.reduceRegions
14 ({
15   collection: zona,
16   reducer: reducers,
17   scale: 20,
18 });
19
20 var featureCollection1 = MeansOfFeatures1.map(function(feet)
21 {
22   var nullfeat1 = ee.Feature(null);
23   return nullfeat1.copyProperties(feet);
24 });
25 var featureCollection2 = MeansOfFeatures2.map(function(feet)
26 {
27   var nullfeat2 = ee.Feature(null);
28   return nullfeat2.copyProperties(feet);
29 });
30 var featureCollection3 = MeansOfFeatures3.map(function(feet)
31 {
32   var nullfeat3 = ee.Feature(null);
33   return nullfeat3.copyProperties(feet);});
34
35 //Exportación a Google Drive
36 Export.table.toDrive
37 ({
38   collection: featureCollection1,
39   description: 'NDVI',
40   folder: 'export_folder_ene'+year,
41   fileFormat: 'CSV'
42 });
43 ...
```

Listado 4.5: Exportación en CSV

Obtención de Precipitaciones

Los datos acumulados de las precipitaciones en nuestra zona van a ser esenciales para nuestro estudio. Al igual que la reflectancia, obtendremos los datos por mes. El satélite usado es ERA5-Land. La ventaja de usar este satélite es que tiene agregados mensuales que se han calculado previamente para facilitar muchas aplicaciones que requieren un acceso fácil y rápido a los datos.

En primer lugar (Listado 4.6), debemos cargar los datos de la precipitación con **ERA5 Land Monthly Aggregated**, que usa la colección de imágenes de precipitación mensual y mostramos la cantidad total de imágenes de precipitación disponible para la evaluación de posibles errores. Por último, convertimos los valores de precipitación a milímetros.

```

1 var pre = ee.ImageCollection('ECMWF/ERA5_LAND/MONTHLY_AGGR').select('
    total_precipitation_sum');
2
3 var count_p = pre.size();
4 print("total_img_precipitaciones", count_p);
5
6 var pre_mm = pre.map(function(img) {
7   return img.multiply(1000).copyProperties(img, ['system:time_start', 'system:time_end']);
8 });
9
10 //Seleccionamos el año en el que queremos obtener los datos.
11 var year = '2023';
12 var months = ['ene', 'feb', 'mar', 'abr', 'may', 'jun', 'jul', 'ago', 'sep', 'oct', 'nov',
13   'dic'];
14
15 //Función para copiar propiedades de una Feature a otra.
16 function copyProperties(feature) {
17   var nullfeat = ee.Feature(null);
18   return nullfeat.copyProperties(feature);}

```

Listado 4.6: Cargar datos de precipitaciones y conversión a milímetros

Ahora, iteramos sobre los meses (enero-diciembre) del año definido (Listado 4.7), se definen en comentarios lo qué hace cada parte del código.

```

1 for (var i = 0; i < 12; i++)
2 {
3   var startdate = ee.Date.fromYMD(parseInt(year, 10), i + 1, 1);
4   var enddate = ee.Date.fromYMD(parseInt(year, 10), i + 1, 1).advance(1, 'month');
5   var pre_month = ee.ImageCollection(pre_mm.filterDate(startdate, enddate));
6   // Todos los datos disponibles para el mes actual
7   var count_p_month = pre_month.size();
8   print("total_img_mes_" + months[i], count_p_month);
9
10  // Hace la sumatoria del mes y selecciona el área de interés
11  var clipped_p_month = pre_month.sum().clip(zona);
12

```

```

13 // Centrado y visualización
14 Map.centerObject(zona, 16);
15 Map.addLayer(clipped_p_month, {'min': 0, 'max': 30, 'palette': "0000ff,32cd32,ffff00,
    ff8c00,ff0000"}, 'Precipitaciones_Mes_' + months[i]);
16 // Media y exportación
17 var reducers = ee.Reducer.mean();
18 var MeansOfFeatures = clipped_p_month.reduceRegions({
19   collection: zona,
20   reducer: reducers,
21   scale: 30,
22 });
23 var featureCollection = MeansOfFeatures.map(copyProperties);
24 Export.table.toDrive({
25   collection: featureCollection,
26   description: 'precipitaciones_' + year + '_' + months[i],
27   folder: 'datos_lluvias_' + year,
28   fileFormat: 'CSV'
29 });
30 }

```

Listado 4.7: Obtención Precipitaciones por mes

Para cada mes, tendremos un archivo en formato CSV que se almacena en la carpeta que hemos indicado.

Obtención Temperaturas

El último dato que vamos a obtener de Google Earth Engine se trata de la temperatura, concretamente la media, máxima y mínima de temperatura diurna y nocturna del mes (LST) del satélite MODIS para una región específica durante un año. Primero debemos cargar las imágenes **MODIS LST** diurnas y nocturnas de la colección '**MODIS/061/MOD11A2**' (Listado 4.8).

```

1 var modisLSTday = ee.ImageCollection('MODIS/061/MOD11A2').select('LST_Day_1km');
2 var modisLSTnight = ee.ImageCollection('MODIS/061/MOD11A2').select('LST_Night_1km');

```

Listado 4.8: Cargar datos MODIS

A continuación, iniciamos el bucle para exportar los datos por cada mes (Listado 4.9).

Primero filtramos las imágenes diurnas y nocturnas por fechas y áreas de interés. Verificamos si hay imágenes disponibles del mes actual. Realizamos una reducción de regiones para obtener estadísticas (máximo, mínimo, media) sobre la región de interés. Exportamos las tablas de estadísticas de temperatura diurna y nocturna como archivos CSV a Google Drive. Cada archivo CSV exportado contiene las estadísticas de temperatura para el mes y tipo de temperatura (día o noche).


```
1 for (var i = 0; i < 12; i++) {
2   var startdate = ee.Date.fromYMD(parseInt(year, 10), i + 1, 1);
3   var enddate = ee.Date.fromYMD(parseInt(year, 10), i + 1, 1).advance(1, 'month');
4
5   // Filtrar por fechas y área de interés
6   var collectionDay = ee.ImageCollection(modLSTday.filterDate(startdate, enddate));
7   var collectionNight = ee.ImageCollection(modLSTnight.filterDate(startdate, enddate));
8
9   // Verificar si hay imágenes disponibles
10  var countDay = collectionDay.size().getInfo();
11  var countNight = collectionNight.size().getInfo();
12
13  if (countDay > 0 && countNight > 0) {
14    // Tomar el promedio y clip
15    var collectionDayMean = collectionDay.mean().clip(area);
16    var collectionNightMean = collectionNight.mean().clip(area);
17
18    // Centrado y visualización
19    Map.centerObject(area, 8);
20
21    Map.addLayer(collectionDayMean, { 'min': -10, 'max': 50, 'palette': "0000ff,32cd32,
    ffff00,ff8c00,ff0000" }, 'Temperatura_promedio_día');
22    Map.addLayer(collectionNightMean, { 'min': -10, 'max': 50, 'palette': "0000ff,32cd32,
    ffff00,ff8c00,ff0000" }, 'Temperatura_promedio_noche');
23
24    // Reducción y exportación
25    var reducers = ee.Reducer.max().combine({
26      reducer2: ee.Reducer.min(),
27      sharedInputs: true
28    }).combine({
29      reducer2: ee.Reducer.mean(),
30      sharedInputs: true
31    });
32
33    var MeansOfFeatures1 = collectionDayMean.reduceRegions({
34      collection: area,
35      reducer: reducers,
36      scale: 30,
37    });
38
39    var MeansOfFeatures2 = collectionNightMean.reduceRegions({
40      collection: area,
41      reducer: reducers,
42      scale: 30,
43    });
44
45    var featureCollection1 = MeansOfFeatures1.map(copyProperties);
46    var featureCollection2 = MeansOfFeatures2.map(copyProperties);
47
48    // Obtener las tres primeras letras del nombre del mes
49    var monthAbbreviation = startdate.format('MMM').getInfo();
50
51    Export.table.toDrive({
52      collection: featureCollection1,
53      description: 'dia_temp_' + monthAbbreviation,
54      folder: 'datosTemperatura' + year,
55      fileFormat: 'CSV'
56    });
```



```

57
58     Export.table.toDrive({
59         collection: featureCollection2,
60         description: 'noche_temp_' + monthAbbreviation,
61         folder: 'datosTemperatura' + year,
62         fileFormat: 'CSV'
63     });
64 } else {
65     print("No hay datos disponibles para el mes " + (i + 1));
66 }
67 }

```

Listado 4.9: Exportación datos temperatura

Obtención de datos de Dron

Para realizar este paso necesitamos el archivo Shapefile de los olivos generados anteriormente, además, necesitaremos los archivos en formato TIF normalizados de los diferentes índices de vegetación a añadir.

En primer lugar, arrastramos el archivo Shapefile a la pantalla al igual que el archivo en formato TIF de los índices de vegetación. En el menú de la derecha buscamos Estadísticas de zona, la capa de entrada sería nuestro Shapefile y la capa ráster el índice de vegetación, en Estadísticas a calcular, seleccionamos lo que necesitamos, en nuestro caso la media. Le damos a ejecutar.

Seleccionamos esta nueva capa y ya podemos pulsar el botón derecho sobre Estadísticas zona1>Exportar>Guardar objetos seleccionados como. Elegimos la Hoja de cálculo y el nombre del archivo. El resultado en el programa es similar al de la Figura 4.15.

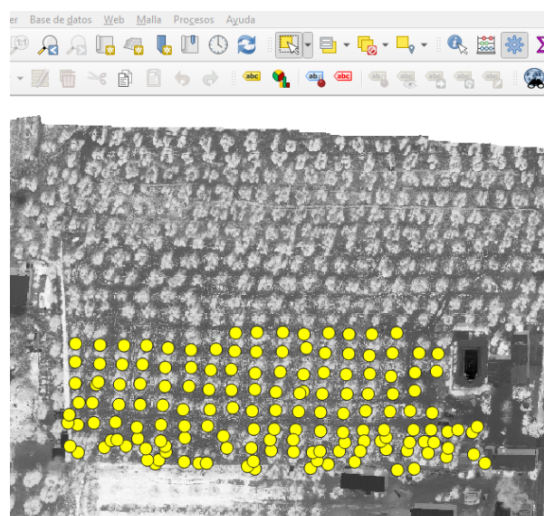


Figura 4.15: Obtención datos de dron - Elaboración Propia

4.3.4. Conversión de datos

Una vez descargados los datos, se debe cambiar su extensión de .csv a .xlsl para una mejor inclusión en nuestra base de datos. Para ello, se ha desarrollado una clase Java que, para cada uno de los índices, elige el que nos interesa y nos lo clasifica por mes. A continuación se explicará con más detalle el uso de dichas clases y si hay que realizar alguna conversión adicional.

Conversión datos precipitaciones

Debido a que los datos de las precipitaciones no es necesario almacenarlos en formato JSON son los más fáciles de convertir. Dentro del programa se ha definido la clase **CSVPrecip** que se encargará de hacer la conversión por nosotros.

ATRIBUTOS

- **Map<String, Map <String, String>.>dataMap:** mapa anidado que utiliza cadenas como clave principal y contiene un mapa interno con cadenas como clave y valores. Almacena los datos procesados de los archivos CSV, donde la clave principal es una concatenación de varios campos y el valor es otro mapa con nombres de archivos CSV y sus respectivos valores medios.
- **Set<String>headers:** almacena los nombres de los archivos CSV procesados. Se utiliza para construir los encabezados en el archivo Excel.

MÉTODOS

- **void processCSVFiles(String folderPath):** recorre los archivos en la carpeta proporcionada (*folderPath*), identifica los archivos CSV y llama al método *processCSV()* para procesar cada archivo.
- **void writeExcelFile(String excelFile, String year):** crea un archivo Excel y escribe los resultados procesados en él. Utiliza la biblioteca Apache POI para manejar archivos Excel (*XSSFWorkbook, Sheet, Row, Cell*).
- **void processCSV(File file):** procesa un archivo CSV específico. Lee los datos del archivo, extrae información relevante y actualiza *dataMap* y *headers*.
- **String getMonthName(String month):** convierte un código de tres letras de mes en su nombre completo.

- **void setCell(Cell cell, String value):** establece el valor de una celda en función de si el valor puede convertirse a un número o no.

Cada una de las ejecuciones por años del apartado anterior genera un documento dentro de la carpeta que hemos declarado en el programa principal. Se ha creado una clase adicional llamada **ExcelCombinator**. Esta clase se utiliza para combinar todos los archivos creados en el apartado anterior en uno final. A continuación se describirán los métodos con mayor detalle.

- **combineExcelFiles(String folderPath, String outputFile):** este método toma la ruta de una carpeta que contiene archivos XLSX y combina su contenido en un solo archivo, cuya ruta se especifica como parámetro. Verifica si hay archivos XLSX en la carpeta especificada, crea un nuevo libro de trabajo y una hoja llamada DatosCombinados, copia los encabezados de la primera hoja del primer archivo al archivo combinado, copia los datos de todas las hojas de todos los archivos al archivo combinado y guarda el archivo combinado en la ruta especificada.
- **copyHeader(File file, Sheet combinedSheet) (privado):** copia los encabezados de la primera hoja del archivo especificado a la hoja combinada.
- **copyData(File file, Sheet combinedSheet) (privado):** copia los datos de todas las hojas del archivo especificado a la hoja combinada.

Desde el programa principal habría que llamar a la primera función mandando la carpeta donde se encuentran los archivos y el nombre del archivo que queremos que genere en la ruta.

Conversión datos temperaturas

Para procesar los datos de la temperatura, necesitamos la media, la máxima y la mínima tanto nocturna como diurna de cada mes. Para ello, se ha creado la clase **CSVPTemperatura** de la siguiente manera:

ATRIBUTOS

- **private Map<String, Map<String, String>.>dataMap:** mapa anidado que utiliza cadenas como clave principal y contiene un mapa interno con cadenas como clave y valores. Almacena los datos procesados de los archivos CSV relacionados

con temperaturas. La clave principal es una concatenación de varios campos (municipio, parcela, polígono, recinto, año, mes), y el valor es otro mapa con nombres de archivos CSV y sus respectivos valores (máximo, medio y mínimo).

- **private Set<String>headers:** conjunto de cadenas que almacena los nombres de los archivos CSV procesados relacionados con temperaturas. Se utiliza para construir los encabezados en el archivo Excel.

MÉTODOS

- **public void processCSVFiles(String folderPath):** recorre los archivos en la carpeta proporcionada (folderPath), identifica los archivos CSV y llama al método *processCSV()* para procesar cada archivo.
- **public void writeExcelFile(String excelFile, String year):** crea un archivo Excel y escribe los resultados procesados relacionados con temperaturas en él. Utiliza la biblioteca Apache POI para manejar archivos Excel (*XSSFWorkbook, Sheet, Row, Cell*).
- **private void processCSV(File file):** procesa un archivo CSV específico relacionado con temperaturas. Lee sus datos, extrae información relevante y actualiza *dataMap* y *headers*. Los índices (como *indiceMunicipio*, *indiceParcela*, etc.) se utilizan para acceder a columnas específicas en el archivo CSV y extraer los datos relacionados con temperaturas. Además, los encabezados y valores se almacenan en el mapa *dataMap* con claves específicas para cada tipo de temperatura (máximo, medio y mínimo). Además, se actualizan los encabezados en el conjunto *headers*.
- **private String getMonthName(String month):** convierte un código de tres letras de mes en su nombre completo. item **private void setCell(Cell cell, String value):** establece el valor de una celda en función de si el valor puede convertirse a un número o no.

Se fusionarán los XLSL llamando a la clase del apartado anterior.

Para convertir los datos *xlsx* a *JSON* se utilizará la herramienta **Pentaho Spoon**. Pentaho es un software de Gestión de Inteligencia Empresarial que permite la mejora de la toma de decisiones, gracias a su capacidad de generar cuadros de mandos o analíticas. Uno de los componentes de Pentaho, es Pentaho Data Integration (PDI) el cual contiene, a su vez, componentes muy interesantes como Spoon.

En el menú de la izquierda aparecen todas las opciones disponibles. Para nuestro caso. Definiremos como entrada un archivo Microsoft Excel, que se encuentra en la carpeta Entrada. Para concatenar los valores de la clave principal, buscamos la opción de Concat fields en la carpeta Transformar. Después, debemos seleccionar los valores que realmente necesitamos para nuestro JSON, usamos la opción Select values de Transformar. Por último, para definir la salida seleccionamos JSON output de la carpeta Salida. Se quedaría como en la Figura 4.16



Figura 4.16: Pasos para transformar XLSL en JSON Pentaho Spoon - Elaboración Propia

1. **Microsoft Excel Input:** seleccionamos el archivo que queremos convertir. En la pestaña Hojas, seleccionamos DatosCombinados. En la pestaña Campos, pulsamos sobre Obtener campos de cabecera. Si previsualizamos las filas, debemos ver todo sin problema.
2. **Concat fields:** para una mejor identificación de la parcela, se concatenarán los campos *Municipio:Poligono:Parcela:Recinto* dándole de nombre ID.
3. **Select values:** pasamos a seleccionar y renombrar valores, en nuestro caso no vamos a renombrar ninguno y a la hora de seleccionar, debemos seleccionar todos incluido el ID del anterior apartado pero eliminando los que hemos concatenado.
4. **JSON output:** elegimos el nombre del bloque, el número de filas que queremos en un bloque y dónde se guardarán.

El resultado, es un archivo como el siguiente 4.17 para todas las filas del XLSL que hemos seleccionado como input.

```

{ } temperatura_20240222_0.json
C: > Users > tic144 > Desktop > PRACTICAS > Transformador > src > main > archivosJSON > JSONTemperaturas > { } temperatura_20240222_0.json > ...
1  [{"data":[{"ID":"32,0:3,0:101,0:3,0"}, {"Año":"2016"}, {"Mes":"Septiembre"}, {"max_dia":"38,4750000000002"},
2  {"mean_dia":"37,590088629290506"}, {"min_dia":"37,16500000000035"}, {"max_noc":"18,77500000000006"},
3  {"mean_noc":"18,72188872851927"}, {"min_noc":"18,63500000000002"}]}]
  
```

Figura 4.17: Resultado Temperatura JSON - Elaboración Propia

Conversión datos índices de vegetación

La conversión de los índices de vegetación es la más complicada ya que tenemos varios y hay que clasificarlos según los meses. Se ha creado la clase **CSVIndices** con las siguientes características:

ATRIBUTOS

- **private Map<String, Map<String, String>.>dataMap:** mapa anidado que utiliza cadenas como clave principal y contiene un mapa interno con cadenas como clave y valores. Almacena los datos procesados de los archivos CSV. La clave principal es una concatenación de varios campos (municipio, parcela, polígono, recinto, año, mes), y el valor es otro mapa con nombres de archivos CSV y sus respectivos valores medios.
- **private Set<String>headers:** conjunto de cadenas que almacena los nombres de los archivos CSV procesados. Se utiliza para construir los encabezados en el archivo Excel.

MÉTODOS

- **public void processCSVFiles(String folderPath):** recorre los archivos en la carpeta proporcionada (folderPath), identifica los archivos CSV y llama al método *processCSV()* para procesar cada archivo.
- **public void writeExcelFile(String excelFile, String year):** crea un archivo Excel y escribe los resultados procesados en él. Utiliza la biblioteca Apache POI para manejar archivos Excel (*XSSFWorkbook, Sheet, Row, Cell*).
- **private void processCSV(File file):** procesa un archivo CSV específico. Se leen los datos del archivo CSV y se extraen los valores de municipio, parcela, polígono, recinto y valor medio. Estos valores se utilizan para construir una clave única en dataMap. El archivo CSV también se utiliza como clave en el conjunto de headers. Los datos se almacenan en la estructura de mapa dataMap para su posterior procesamiento y escritura en el archivo Excel.
- **private String getMonthName(String month):** convierte un código de tres letras de mes en su nombre completo.
- **private void setCell(Cell cell, String value):** establece el valor de una celda en función de si el valor puede convertirse a un número o no.

Para la fusión de XLSL se seguiría la misma metodología que la clase anterior.

Para convertir los datos de los índices de vegetación a JSON se seguirían los mismos pasos que en el apartado anterior pero cambiando los campos. Elegimos el fichero resultadoINDICES.xlsx, la hoja y los campos correspondientes. Elegimos los campos que queremos concatenar, en este caso serían los mismos. Seleccionamos los valores, debemos darle a “Obtener campos a seleccionar” para obtener los nuevos campos de la tabla y debemos eliminar los otros. En JSON output, hacemos lo mismo en la pestaña fields. El resultado sería como el de la Figura 4.18

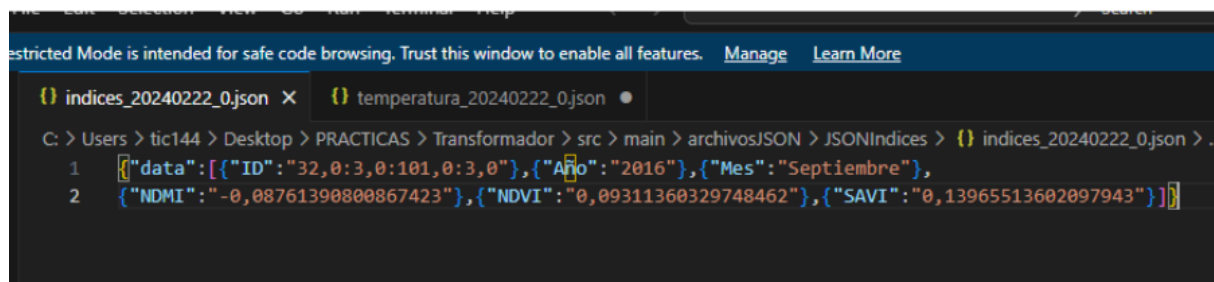


Figura 4.18: Resultado Índices JSON - Elaboración Propia

4.4. Segunda iteración: Diseño de la Base de Datos

4.4.1. Introducción

En la segunda iteración del proyecto nos vamos a centrar en el diseño de la estructura en la que se van a almacenar los datos. Para realizar lo anteriormente mencionado, se partió de un diseño previo aconsejado por el grupo de investigación que aparece en la Figura 4.19. En un futuro, se podrá conectar la Base de Datos modificada que en este proyecto independiente se genera con la del grupo de investigación, de forma que esta se encuentre completa.

El código Java necesario para esta iteración, se puede encontrar en el siguiente repositorio de [GitHub](#).

Estudios de alternativas y viabilidad

En esta iteración, la principal decisión a tomar fue la herramienta de gestión para la Base de Datos. Las dos herramientas que se pensaron fueron Oracle y MongoDB.



Figura 4.19: Base de Datos Inicial - Elaboración propia

Finalmente, se eligió Oracle puesto que tiene un robusto soporte para transacciones complejas y una gran consistencia de datos, además, necesitábamos una base de datos estructurada. Por otro lado, la Base de Datos inicial del grupo de investigación estaba implementada en Oracle, por tanto, para luego ser completada sería más fácil.

Tecnologías Usadas

A continuación se explicarán todas las tecnologías usadas:

■ Lenguajes de programación

- **SQL (Structured Query Language):** lenguaje estándar para la gestión y manipulación de bases de datos relacionales. Se utiliza para realizar consultas, insertar, actualizar y eliminar datos, así como para definir y modificar la estructura de las bases de datos. Se ha usado para crear la Base de Datos implementada en Oracle.

■ Entornos de desarrollo:

- **SQLDeveloper:** para la gestión de la base de datos se usó SQL Developer. Este es una herramienta de desarrollo de bases de datos proporcionada por Oracle para trabajar específicamente con esas bases de datos. Algunas de sus características clave son:
 - **Editor SQL avanzado:** con resaltado de sintaxis, autocompletado de código, formateo de código y navegación rápida por objetos de base de datos.
 - **Explorador de objetos de base de datos:** permite explorar la estructura de la base de datos, ver tablas, vistas, procedimientos almacenados, funciones y otros objetos, así como modificar y administrar estos objetos directamente desde la interfaz de usuario.
 - **Generación de informes y consultas:** permite ejecutar consultas SQL y generar informes personalizados basados en los resultados de las consultas. También incluye herramientas para la visualización de datos y la creación de gráficos.
 - **Herramientas de administración de bases de datos:** Ofrece herramientas como la gestión de usuarios, roles y privilegios, la monitorización del rendimiento de la base de datos y la gestión de copias de seguridad y restauración.

Se ha elegido Oracle Database por muchas razones, entre ellas: [36]

- **Rendimiento y escalabilidad probados:** tiene un historial probado de rendimiento y escalabilidad, lo que lo convierte en una opción confiable para aplicaciones de misión crítica.
- **Amplia gama de funciones:** ofrece una amplia gama de funciones para satisfacer las necesidades de las empresas de todos los tamaños.
- **Seguridad sólida:** ofrece una amplia gama de funciones de seguridad para proteger sus datos de accesos no autorizados.
- **Opciones de implementación flexibles:** se puede implementar en una variedad de entornos, incluidos locales, en la nube e híbridos.
- **Amplio ecosistema de socios:** existe un amplio ecosistema de socios de Oracle que ofrecen soporte, capacitación y servicios para Oracle Database.
- **Oracle Spatial:** como requerimos la gestión de datos geográficos, Oracle nos proporciona soporte mediante Oracle Spatial. Agrega capacidades avanzadas para el manejo de datos espaciales. Permite almacenar, consultar y analizar datos geoespaciales dentro de la base de datos relacional de Oracle. Se puede trabajar con datos como puntos, líneas y polígonos.

4.4.2. Diseño de la Base de Datos

Nuestro proyecto contará únicamente con una Base de Datos implementada en Oracle, que se conectará al Backend de la aplicación web. En la figura 4.19 podemos apreciar un total de veintiséis tablas. No todas son necesarias para este proyecto, por ello, nos vamos a centrar en Cosecha, Finca, Historico_Datos, Historico_Finca, Municipio, Objeto, Provincia y Zona.

En el apartado anterior, se especificó que se debían añadir datos no estructurados en la Base de datos. Después de un exhaustivo análisis, se concluyó que lo más oportuno era añadir datos de tipo BLOB, que almacenarán JSON, no siendo necesaria la estructuración de ellos. Por ejemplo, si se hace un vuelo con dron, puede ser que el dron no tenga bandas para detectar determinados índices de vegetación, pues por cada entrada de la base de datos podría haber distintos.

- Dentro de la tabla **Municipio**, se introdujo PROVINCIA_CODIGO_PROVINCIA como clave primaria junto con el código del municipio. Este cambio se verá reflejado en la tabla **Zona**, cuya clave principal ahora la forma UBICACION, MUNICIPIO_CODIGO y CODIGO_PROVINCIA. De igual forma, la clave primaria de la tabla **Finca** será MUNICIPIO_CODIGO, PROVINCIA_CODIGO, POLIGONO, PARCELA y RECINTO ya que se añade una nueva relación con **MUNICIPIO**. Se añade este atributo, de igual manera, en la tabla **OBJETO e HISTORICO_DATOS**.
- Por otro lado, estimé conveniente eliminar la relación de la tabla cosecha con municipio ya que podríamos obtener esos datos desde **FINCA** y serían redundantes.
- Por otro lado, vemos que existe la tabla **HISTORICO_DATOS** que almacena secuencias temporales de diferentes objetos pero esta tabla no existe a nivel de finca, de la cuál podemos obtener también datos temporales. Por esta razón, se crea una nueva tabla llamada **HISTORICO_FINCA**.

Una vez explicados los cambios, pasaremos a explicar cada uno de los elementos de la base de datos. Además, al final de este apartado, se especificará el Modelo Entidad-Relación, un diagrama que nos indica cómo se organiza la información almacenada en una Base de Datos.

Entidades Almacenadas

Provincia

En esta tabla (Figura 4.20), se representarán las provincias de las que proceden las fincas y zonas.

PROVINCIA		
ATRIBUTO	TIPO	COMENTARIOS
CODIGO_PROVINCIA	NUMBER	Clave Primaria, no nulo.
NOMBRE_PROVINCIA	VARCHAR2	No nulo

Figura 4.20: Tabla Provincia - Elaboración Propia

- **CODIGO_PROVINCIA:** código de la provincia en el Sigpac, la identifica de manera única.
- **NOMBRE_PROVINCIA:** nombre de la provincia.

Municipio

En esta tabla (Figura 4.21), se representarán los municipios de los que proceden las fincas y zonas.

MUNICIPIO		
ATRIBUTO	TIPO	COMENTARIOS
CODIGO_MUNICIPIO	NUMBER	Clave Primaria, no nulo.
PROVINCIA_CODIGO_PROVINCIA	NUMBER	Clave foránea.
NOMBRE	VARCHAR2	-

Figura 4.21: Tabla Municipio - Elaboración Propia

- **CODIGO_MUNICIPIO:** código del municipio que lo identifica de manera única.
- **PROVINCIA_CODIGO_PROVINCIA:** clave foránea que hace referencia al código de la tabla Provincia, se encuentra también en la clave primaria.
- **NOMBRE:** nombre del municipio.

Zona

En esta tabla (Figura 4.22), se representa el término abstracto de zona, esta puede hacer referencia a un conjunto de fincas, a un bosque o a cualquier agrupación que podemos pensar.

ZONA		
ATRIBUTO	TIPO	COMENTARIOS
UBICACION	VARCHAR2	Clave Primaria
LONGITUD	FLOAT	-
LATITUD	FLOAT	-
DESCRIPCION	VARCHAR2	-
MUNICIPIO_CODIGO	NUMBER	Clave Foránea
PROVINCIA_CODIGO	NUMBER	Clave Foránea
RADIO	FLOAT	-

Figura 4.22: Tabla Zona - Elaboración Propia

- **UBICACION:** nombre de la ubicación, es la clave primaria.
- **LONGITUD:** longitud de la ubicación expresadas en coordenadas decimales.
- **LATITUD:** latitud de la ubicación expresadas en coordenadas decimales.
- **DESCRIPCION:** descripción de la zona.
- **MUNICIPIO_CODIGO:** clave foránea que hace referencia al municipio donde se encuentra, se encuentra también en la clave primaria.
- **PROVINCIA_CODIGO:** clave foránea que hace referencia a la provincia donde se encuentra, pertenece también en la clave primaria.
- **RADIO:** extensión de la zona.

Finca

En esta tabla (Figura 4.23), se representan las fincas, están identificadas mediante su código SIGPAC que se mencionó en el Apartado 1.5.

FINCA		
ATRIBUTO	TIPO	COMENTARIOS
AREA	FLOAT	-
POLIGONO	NUMBER	Clave Primaria
PARCELA	NUMBER	Clave Primaria
RECINTO	NUMBER	Clave Primaria
AÑO_SIGPAC	NUMBER	-
ZONA_UBICACION	VARCHAR2	Clave Foránea
MUNICIPIO_CODIGO	NUMBER	Clave Foránea
PROVINCIA_CODIGO	NUMBER	Clave Foránea

Figura 4.23: Tabla Finca - Elaboración Propia

- **AREA:** área de la finca.
- **POLIGONO:** dentro de su código SIGPAC, representa el polígono. Se encuentra en la clave primaria.
- **PARCELA:** dentro de su código SIGPAC, representa la parcela. Se encuentra en la clave primaria.
- **RECINTO:** dentro de su código SIGPAC, representa el recinto. Se encuentra en la clave primaria.
- **AÑO_SIGPAC:** año en el que se inscribió en el SIGPAC.
- **ZONA_UBICACION:** clave foránea que hace referencia a la ubicación de la zona donde está.
- **MUNICIPIO_CODIGO:** clave foránea que también pertenece a la primaria representando el código del municipio donde se encuentra.
- **PROVINCIA_CODIGO:** clave foránea que también pertenece a la primaria representando el código de la provincia donde se encuentra.

Cosecha

En esta tabla (Figura 4.22), se representa la información de las cosechas que han podido tener las fincas mencionadas anteriormente.

COSECHA		
ATRIBUTO	TIPO	COMENTARIOS
ID_COSECHA	NUMBER	Clave Primaria
CAMPAÑA	VARCHAR2	-
MUNICIPIO_CODIGO	NUMBER	Clave Foránea
PROVINCIA_CODIGO	NUMBER	Clave Foránea
ACEITUNAS	FLOAT	-
ACEITE	FLOAT	-
FINCA_POLIGONO	NUMBER	Clave Foránea
FINCA_PARCELA	NUMBER	Clave Foránea
FINCA_RECINTO	NUMBER	Clave Foránea
ZONA_UBICACION	VARCHAR2	Clave Foránea

Figura 4.24: Tabla Cosecha - Elaboración Propia

- **ID_COSECHA:** clave primaria de la tabla, representa un id único.
- **CAMPAÑA:** año de la campaña de la cosecha.
- **MUNICIPIO_CODIGO:** clave foránea que representa el municipio donde se encuentra la finca.
- **PROVINCIA_CODIGO:** clave foránea que representa la provincia donde se encuentra la finca.
- **ACEITUNAS:** kilogramos de aceitunas recogidas en la campaña.
- **ACEITE:** kilogramos de aceite producidos en la campaña.
- **FINCA_POLIGONO:** clave foránea que representa el polígono donde se encuentra la finca.
- **FINCA_PARCELA:** clave foránea que representa la parcela donde se encuentra la finca.
- **FINCA_RECINTO:** clave foránea que representa el recinto donde se encuentra la finca.
- **ZONA_UBICACION:** clave foránea que representa la ubicación de la zona donde se encuentra la finca.

Historico_Finca

En esta tabla (Figura 4.25), se representa la información almacenada para una fecha de la finca, en ella puede haber datos de reflectancia, precipitaciones o temperatura, para cada uno de los históricos hay que indicar de dónde proceden los datos.

HISTORICO_FINCA		
ATRIBUTO	TIPO	COMENTARIOS
FECHA	DATE	Clave Primaria
MUNICIPIO_CODIGO	NUMBER	Clave Foránea
PROVINCIA_CODIGO	NUMBER	Clave Foránea
POLIGONO	NUMBER	Clave Foránea
PARCELA	NUMBER	Clave Foránea
RECINTO	NUMBER	Clave Foránea
REFLECTANCIA	BLOB	-
LLUVIA	NUMBER	-
TEMPERATURA	BLOB	-
NOMBRE_FUENTE	VARCHAR2	-
TIPO_FUENTE	VARCHAR2	-
ZONA_UBICACION	VARCHAR2	Clave Foránea

Figura 4.25: Tabla Histórico Finca - Elaboración propia

- **FECHA:** fecha en la que se obtuvo el histórico, pertenece a la clave primaria.
- **MUNICIPIO_CODIGO:** municipio donde se encuentra la finca de la que es el histórico, clave foránea y principal.
- **PROVINCIA_CODIGO:** provincia donde se encuentra la finca de la que es el histórico, clave foránea y principal.
- **POLIGONO:** polígono de la finca de la que es el histórico, clave foránea y principal.
- **PARCELA:** parcela de la finca de la que es el histórico, clave foránea y principal.
- **RECINTO:** recinto de la finca de la que es el histórico, clave foránea y principal.
- **REFLECTANCIA:** JSON donde se representarán todos los datos de reflectancia del histórico.

OBJETO		
ATRIBUTO	TIPO	COMENTARIOS
ID_OBJETO	NUMBER	Clave Primaria
ZONA_UBICACION	VARCHAR2	Clave Foránea
ZONA_MUNICIPIO_CODIGO	NUMBER	Clave Foránea
ZONA_PROVINCIA_CODIGO	NUMBER	Clave Foránea
POLIGONO_ENVOLVENTE	SDO_GEOMETRY	Not Null
TIPO_OBJETO	VARCHAR2	-
PUNTO_MEDIO	SDO_GEOMETRY	Not Null

Figura 4.26: Tabla Objeto - Elaboración propia

- **LLUVIA:** atributo para presentar el acumulado de la lluvia en la fecha.
- **TEMPERATURA:** JSON para representar las temperaturas medias, máximas y mínimas alcanzadas en la fecha.
- **NOMBRE_FUENTE:** nombre de la fuente donde se obtuvieron los datos.
- **TIPO_FUENTE:** tipo de la fuente donde se obtuvieron los datos, por ejemplo, satélite o dron.
- **ZONA_UBICACION:** clave foránea que representa la ubicación donde se encuentra la finca de la que es el histórico.

Objeto

En esta tabla (Figura 4.26), se representa el término abstracto objeto, en nuestro caso, se corresponden a los olivos, pero si la zona, por ejemplo, es un bosque, se podrían representar cada uno de los árboles que lo componen.

- **ID_OBJETO:** clave primaria de la tabla, el id es único.
- **ZONA_UBICACION:** clave foránea que representa la zona donde se encuentra.
- **ZONA_MUNICIPIO_CODIGO:** clave foránea que representa el municipio donde se encuentra la zona del objeto.
- **ZONA_PROVINCIA_CODIGO:** clave foránea que representa la provincia donde se encuentra la zona del objeto.

- **POLIGONO_ENVOLVENTE:** se trata de un atributo del tipo SDO_GEOMETRY para representar el polígono envolvente del objeto.
- **TIPO_OBJETO:** tipo del objeto que se representa.
- **PUNTO_MEDIO:** atributo del tipo SDO_GEOMETRY para representar el punto medio del objeto, para su representación.

Historico_Datos

Al igual que se ha hecho con finca, podemos almacenar información específica del objeto para una determinada fecha (Figura 4.27).

HISTORICO_DATOS		
ATRIBUTO	TIPO	COMENTARIOS
FECHA	DATE	Clave Principal
VOLUMEN	FLOAT	-
REFLECTANCIA	BLOB	Not Null
ID_OBJETO	NUMBER	Clave Foránea
NOMBRE_FUENTE	VARCHAR2	-
TIPO_FUENTE	VARCHAR2	-

Figura 4.27: Tabla Histórico Datos - Elaboración Propia

- **FECHA:** clave primaria, fecha en la que se obtuvieron los datos.
- **VOLUMEN:** volumen que pueda tener el objeto en la fecha.
- **REFLECTANCIA:** JSON para representar los índices de vegetación en la fecha.
- **ID_OBJETO:** clave primaria que hace referencia al objeto del que provienen los datos.
- **NOMBRE_FUENTE:** nombre de la fuente donde se obtuvieron los datos.
- **TIPO_FUENTE:** tipo de la fuente donde se obtuvieron los datos.

Una vez explicadas todas las tablas, en la Figura 4.28 podemos ver el Diagrama Entidad-Relación.

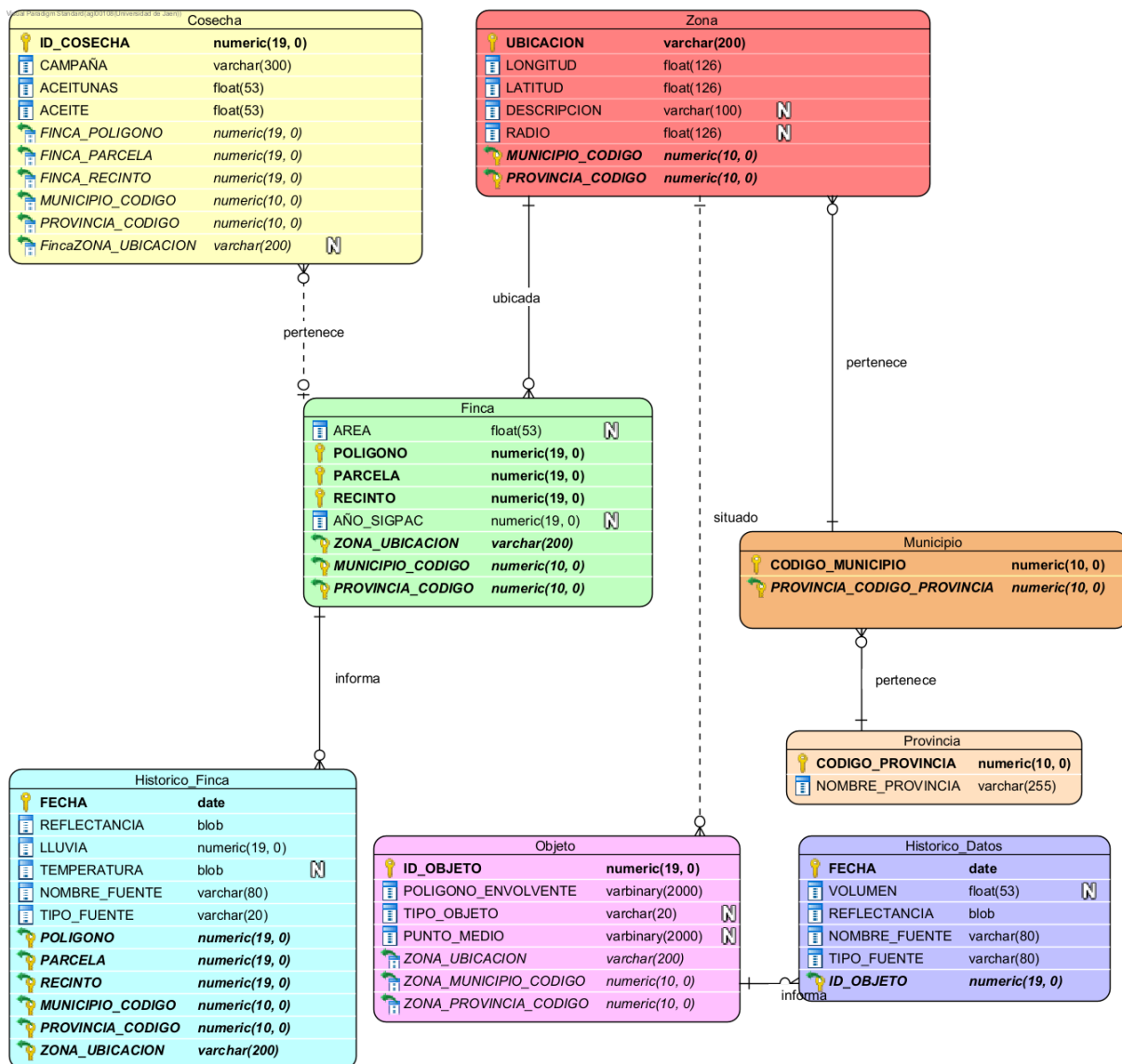


Figura 4.28: Diagrama Entidad Relación - Elaboración Propia

Inserción Histórico Finca en la Base de Datos

Una vez que tenemos todos los campos para insertar en nuestra base de datos, debemos ver la forma en la que podemos automatizar el insertado de las tuplas. Por un lado, tendríamos el archivo XLSL de las precipitaciones, que no necesitan su conversión a archivo JSON, de aquí obtendremos todos los atributos para realizar el INSERT INTO menos la reflectancia y la temperatura, que se corresponden con archivos JSON separados.

Para ello, se ha creado la última clase llamada **SQLConvertor**, que procesa un archivo Excel y sus JSON correspondientes y genera sentencias INSERT INTO en formato sql para directamente insertarlas en la base de datos.

Esta tiene dos atributos estáticos:

- **lastRowIndices:** índice del último archivo procesado de índices. Inicializado en 0. Se ha incluido puesto que puede haber meses que no se tengan datos de los índices porque cuando estamos obteniendo los datos de Google Earth Engine, limitamos el número de nubes para obtener un resultado válido. Si ese mes el satélite ha pasado cuando había muchas nubes, debemos descartar el resultado.
- **encontrado:** boolean que indica si se ha encontrado el dato. Inicializado en true. Lo usamos para cuando falte un archivo de índices por la razón explicada en el apartado anterior, cambiamos a false y empezamos a tomar, en vez de la fila por la que vamos, la última dónde no se encontraron los índices y así seguir en orden.

En cuanto a los métodos, estos son:

- **generarInsercionDesdeExcel:** toma la ruta del archivo Excel como parámetro, abre el archivo Excel y procesa cada fila (a partir de la segunda fila ya que la primera se trata de los encabezados). Obtiene los datos de cada columna (provincia, municipio, polígono, etc.) que necesitaremos y llama al método *generarInsertInto* para construir las sentencias SQL INSERT INTO correspondientes, mandando todos los parámetros necesarios.
- **generarInsertInto:** toma los datos obtenidos de la fila del archivo Excel y los utiliza para construir una sentencia SQL INSERT INTO. Hay que convertir la fecha al formato adecuado (yyyy-MM-dd). Además, llama al método *obtenerContenidoJSON* para obtener el contenido JSON correspondiente tanto de índices de vegetación como de temperaturas. Si al llamar a este método nos devuelve un valor nulo, se insertaría un 'NULL' en esa parte. Escribe la sentencia SQL resultante en un archivo .sql.
- **obtenerContenidoJSON:** construye la ruta del archivo JSON a partir de los parámetros dados. Tras ello, debemos tener en cuenta el archivo a procesar, si el atributo encontrado se encuentra a true, se obtienen los datos en orden de las distintas carpetas. Si no, se debe tener en cuenta el atributo *lastRowIndices* ya que el orden se ha perdido puesto que faltaría un JSON de índices. Para las temperaturas no habría problema, ya que se obtienen todos los datos del satélite. Después, usa la biblioteca Jackson (*ObjectMapper*) para leer el contenido del archivo JSON. Compara los datos del archivo JSON con los datos proporcionados

para verificar si coinciden. Devuelve el contenido del JSON si hay coincidencia, de lo contrario, retorna *null*.

- **obtenerNumeroMes:** se trata de un método auxiliar que convierte el nombre del mes a su número correspondiente.

Al final, tendríamos un archivo SQL listo para insertar en nuestra base de datos como se puede ver en el Listado 4.10.

```

1 INSERT INTO HISTORICO_FINCA
2   (FECHA, PROVINCIA_CODIGO, MUNICIPIO_CODIGO, ZONA_UBICACION, POLIGONO, PARCELA,
3    RECINTO, LLUVIA, REFLECTANCIA, TEMPERATURA)
4 VALUES
5   (TO_DATE('2020-07-01', 'YYYY-MM-DD'), 23, 5, 'J2', 7, 325, 2, 5.271451819336193,
6    utl_raw.cast_to_raw('{"data":[{"ID":"23:5:7:325:2"},"Año":"2020"},"Mes":"Julio
7    "},{"NDMI":"-0,04585261591114742"},"NDVI":"0,12602839985147885"},"SAVI
    ":"0,18902424965317477"}]}'), --
    Reflectancia
6    utl_raw.cast_to_raw('{"data":[{"ID":"23:5:7:325:2"},"Año":"2020"},"Mes":"Julio
    "},{"max_dia":"44,315000000000026"},"mean_dia":"44,30660295451603"},"min_dia
    ":"43,715000000000002"},"max_noc":"26,545000000000003"},"mean_noc
    ":"26,53954192043545"},"min_noc":"26,1550000000000044"}]}') --
7    Temperatura
);

```

Listado 4.10: Ejemplo de inserción en Historico Finca

Inserción Histórico Datos en la Base de Datos

Una vez obtenidos los datos en la iteración anterior y teniendo creadas las tablas, es momento de introducirlos en nuestra Base de Datos. Como se dijo anteriormente, tenemos dos fuentes de datos, satelital y mediante dron. Los datos de drones, que son muy escasos, representan valores más acertados que los de satélite, ya que estos dependen de la resolución de los satélites que se están usando. Para poder distinguirlos, tenemos *nombreFuente* y *tipoFuente* entre los atributos.

Inserción satelital

En este apartado se explicará cómo conseguir las sentencias SQL necesarias para poder introducir en la Base de Datos todos los históricos de los objetos. Debemos pensar que, por un lado, tenemos los CSV con los datos de los olivos descargados en la iteración primera y por otro el SQL con los datos de los mismos. Como el ID de los olivos, que es la clave foránea, es generado automáticamente por SQLDeveloper, debemos descargarnos los datos que se han insertado.

En primer lugar, dentro de la clase descrita anteriormente (*CSVIndicesO*), se ha generado la función `public void readOlivelds(String sqlFile)`, esta mapeará los índices de cada olivo junto con las coordenadas x e y del centro en un mapa.

Tras ella, tenemos `public void processCSVFiles(String folderPath)`, que procesa cada uno de los archivos CSV generados por Google Earth Engine, de igual forma que a nivel de parcela. Llamaría a `private void processCSV(File file)`, que obtiene de cada uno la coordenada x e y del centro del olivo junto con el índice de vegetación asociado. Como diferencia, cabe destacar que se mira si el mapa del id de los olivos contiene el punto medio, ya que si recordamos, hemos hecho un recorte de los olivos que nos proporcionaba GEU, lo añade a la clave para luego poder añadirlo en el archivo XLSL para poder identificarlo e insertarlo en la base de datos.

Tenemos, de igual modo, la función `public void writeExcelFile(String excelFile)`, que, a partir de las claves almacenadas anteriormente de los archivos en formato csv, nos escribe todos los datos en un excel final.

Por último, tenemos la función `public void generateSQLFromExcel(String excelFilePath, String jsonFolderPath, String sqlFilePath, String nombreFuente, String tipoFuente)`, a esta le debemos mandar el XLSL generado en el anterior paso, la carpeta donde tenemos todos los JSON generados con Pentaho Spoon de los índices de vegetación, el fichero donde queremos que se añadan las sentencias de inserción y el nombre y el tipo de la fuente donde se obtuvieron los datos.

Esta función, obtiene de la clave todos los datos necesarios, crea la fecha, que siempre se corresponde con el primer día del mes de cada mes y año generados, busca y procesa el archivo json correspondiente. Destacar que deben estar en orden los archivos json y las filas del excel final, no debería haber problema si se sigue el orden de los pasos. Por último, construye la sentencia de inserción, acabando nuestro proceso de tratado de datos del olivo con un resultado similar al Listado 4.11.

```

1 INSERT INTO HISTORICO_DATOS (FECHA, VOLUMEN, REFLECTANCIA, ID_OBJETO, NOMBRE_FUENTE,
  TIPO_FUENTE)
2 VALUES (TO_DATE('2017-3-01', 'YYYY-MM-DD'), NULL, utl_raw.cast_to_raw('{"data": [{"ID
  ": "772"}, {"Año": "2017"}, {"Mes": "Marzo"}, {"NDMI": "0,11315944045782089"}, {"NDVI
  ": "0,345661461353302"}, {"SAVI": "0,5184307098388672"}]}'), 772, 'Sentinel-2', 'Satelite')
  ;

```

Listado 4.11: Ejemplo de inserción en Histórico Datos

Inserción datos dron

En primer lugar, necesitaremos un listado de los Olivos almacenados, ya que necesitamos obtener su ID a partir de los puntos medios. Para realizar todas las operaciones se ha creado la clase **SQLConvertorDron**, a continuación explicaré cada uno de los métodos que contiene:

La clase **SQLConvertorDron** es una clase en Java que se utiliza para procesar datos descargados de un dron sobre índices de vegetación y convertirlos a un formato XLSL y su posterior archivo de inserción en la base de datos.

ATRIBUTOS:

- **carpeta:** Es una cadena que representa la ruta de la carpeta donde se encuentran los archivos a procesar descargados del Dron mediante QGis.
- **olivelds:** Es un mapa que almacena los IDs de los olivos. Las claves son cadenas que representan las coordenadas de los olivos y los valores son los IDs de los olivos.

MÉTODOS

- **procesarArchivos(String rutaGuardar):** este método procesa los archivos en la carpeta especificada y guarda los datos procesados en la ruta proporcionada. Se necesita pasar la ruta donde guardar el archivo Excel final. Llama al método procesarHoja y escribirArchivoFinal, ámbos privados.
- **procesarHoja(Sheet hoja, String nombreArchivo, Map<String, List<Double>.>datosPorPar):** este método privado procesa una hoja de un archivo .xlsx y almacena los datos en el mapa *datosPorPar*. Para ello, obtiene los puntos medios de las celdas y mira si están almacenados en el mapa, si es así sólo añadiría la tercera columna que corresponde con el índice de vegetación especificado y si no, haría lo anterior previamente añadiendo los datos del par.
- **escribirArchivoFinal(Map<String, List<Double>.>datosPorPar, String rutaGuardar):** este método privado escribe los datos procesados en un archivo .xlsx en la ruta proporcionada. Una vez obtenidos los datos en el mapa *datosPorPar*, nos queda escribirlos correctamente en el Excel final para poder procesarlos con Pentaho Spoon.

- **readOlivelds(String sqlFile):** este método, similar al explicado anteriormente, guarda en un mapa el ID del Olivo junto con sus puntos medios, para el correcto parcheado de datos. Como los datos en QGis se descargan con más precisión, se ha limitado el número de decimales a 3.
- **generarSQLParaDron(String excelFilePath, String jsonFolderPath, String sqlFilePath, String nombreFuente, String tipoFuente):** método similar al generado para los datos de satélites con la diferencia de que cambiaría el tipo de fuente.
- **String readJSON(String jsonFilePath):** método auxiliar para procesa datos de un JSON.

Recaltar en este apartado que se ha seguido la misma metodología para obtener los datos en formato JSON en el programa Pentaho Spoon de la misma manera que para los datos de satélite. El nuevo archivo SQL de inserción tendrá formato similar a la Figura 4.11.

También debemos añadir los datos para Histórico Finca, estos se corresponden con la media de todos los olivos que acabamos de insertar. Para ello, en la clase *SQL-ConvertorDron* se ha generado la funcion *public void generarMedia(String mediaPath, Integer provincia, Integer municipio, String zona, Integer poligono, Integer parcela, Integer recinto, String tipoFuente, String nombreFuente)*. En ella, se le pasa todos los datos necesarios sobre la Finca y genera la sentencia SQL de Histórico Finca pero con la media de los índices de sus olivos y ya directamente la podríamos almacenar en la Base de Datos.

4.5. Tercera iteración: Arquitectura del sistema y pruebas

4.5.1. Introducción

En la tercera iteración de nuestro proyecto, comenzaremos con el desarrollo Backend de la aplicación. Este, como se mencionó en el apartado de tecnologías, se realizó con **Spring Framework**, dividiendo el desarrollo en la capa de dominio, persistencia e implementación de un servicio REST.

El código para esta iteración y la siguiente se encuentra disponible en el siguiente repositorio de [GitHub](#).

Estudios de alternativas y viabilidad

A la hora de diseñar el backend de una aplicación web, existen numerosos Frameworks como Django, Flask, Ruby on Rails, Express.js, Laravel o Spring. Se eligió Spring ya que proporciona un ecosistema robusto y completo para el desarrollo de aplicaciones empresariales. Ofrece una gran flexibilidad y una amplia gama de funcionalidades integradas, como gestión de transacciones, seguridad, y conectividad con bases de datos. Además, tenía soporte para Oracle y para los datos geométricos que queríamos almacenar.

Tecnologías Usadas

Destacar que el lenguaje de programación fue Java y el entorno de programación IntelliJ, explicados anteriormente en la Iteración Primera [4.3](#).

■ Spring Framework



Figura 4.29: Frameworks Java

Existen muchos Frameworks Java como se puede ver en la Figura [4.29](#), el elegido finalmente fue Spring. Este, es una plataforma en Java que proporciona un completo conjunto de herramientas de infraestructura destinadas al desarrollo de aplicaciones empresariales en este lenguaje. Con Spring, se delega el manejo de la infraestructura, permitiendo al desarrollador focalizarse plenamente en el

diseño y la implementación de la lógica de la aplicación. Surgió como una alternativa ligera a la compleja plataforma J2EE, ganando mucha popularidad [37].

Una de las características más destacadas de este Framework es su capacidad para construir aplicaciones utilizando el paradigma de los "Plain Old Java Objects"(POJOs) [38], lo que implica que se pueden crear objetos Java estándar sin necesidad de extender clases o implementar interfaces específicas del Framework. Además, hay que destacar la inyección de dependencias, gestión de transacciones, soporte para pruebas automatizadas y el soporte orientado a aspectos de programación.

Spring tiene Frameworks complementarios como [37]:

- **Spring MVC:** módulo de Spring para desarrollar rápidamente aplicaciones web. Se basa en el Modelo-Vista-Controlador.
- **Spring Security:** módulo para la inserción de funcionalidades para autenticación y autorización.
- **Spring Data:** debido al auge de nuevas tecnologías como almacenamiento NoSQL se creó este módulo para base de datos y servicios en la nube.
- **Spring Web Services:** creación de servicios Web basados en SOAP.
- **Spring Web Flow:** extensión para la implementación de flujos (Wizards) de pantallas.



Figura 4.30: Logo Spring Framework

Spring Boot

Java Spring Boot es una herramienta de código abierto que facilita el uso de marcos basados en Java para crear microservicios y aplicaciones web. Para cualquier definición de Spring Boot, la conversación tiene que comenzar con Java, uno de los lenguajes de desarrollo y plataformas informáticas más populares y ampliamente usados para el desarrollo de aplicaciones [39].

Nos podríamos preguntar, por tanto, ¿cuál es la diferencia entre Spring Framework y Spring Boot? Para comprender las diferencias, primero se deben definir algunos aspectos clave.

Microservicios: se trata de un enfoque arquitectónico que se centra en la creación de una aplicación a través de un conjunto de servicios independientes, escalables, colaborativos y evolutivos que se adaptan a ecosistemas complejos. Los microservicios surgieron para superar las limitaciones de los sistemas monolíticos, que se distinguen por mantener los servicios como una sola unidad lógica y utilizar los mismos recursos computacionales [40].

Inversión del control (IoC): se trata de una estrategia que consiste en postergar la resolución de dependencias al tiempo de ejecución, en lugar de decidirla en tiempo de compilación. Requiere un framework que resuelva las conexiones durante la fase de ejecución. En el caso de Spring Framework se trata del contenedor IoC. Se hace de forma declarativa, evitando secuencias programáticas de comandos explícitos.

Inserción de dependencias: la inversión de control (IoC) se logra mediante la técnica de diseño de inserción de dependencias (DI). En la programación orientada a objetos como Java, las dependencias son objetos que dependen de otros objetos. El cliente es el objeto receptor o dependiente, y el servicio es el objeto del que depende el cliente. Por lo tanto, el servicio al cliente inserta la dependencia mediante un código llamado inyector [39].

Dado que Spring Boot se basa en Spring Framework, es una situación en la que es posible beneficiarse de ambos ya que van de la mano.

Spring Boot proporciona una ruta de acceso más fácil y rápida para configurar, establecer y ejecutar aplicaciones. Elimina el trabajo pesado de configuración necesario para configurar la mayoría de las aplicaciones basadas en Spring. Los desarrolladores pueden entrar directamente y usar Spring Boot sin tener que aprender el marco de Spring subyacente.

Ventajas e inconvenientes

Spring Framework no es la única tecnología de desarrollo web en Java, existen otras como JavServer Faces (JSF), Apache Struts o Play Framework. Spring ha ganado una gran popularidad y es considerado como uno de los mejores frameworks por distintas razones:

- **Manejo de transacciones:** Spring ofrece un amplio soporte para el manejo de transacciones. Se incluye el soporte para Java Transaction API (JTA), el manejo de transacciones en capas de acceso a datos usando tecnologías como el mapeo objeto-relacional (ORM) (Hibernate, Java Data Objects o JDBC).
- **Persistencia de entidades:** Spring usa implementaciones ORM de terceros como Hibernate, Ibatis, JDO, OJB.

- **Programación Orientada a Aspectos:** Spring permite usar archivos XML de configuración, programación mediante la API y mediante un estándar JSR.
- **Seguridad:** Spring aporta integración con la solución de código abierto Acegi (soporta seguridad declarativa basada en el uso de IoC y AOP).
- **Flexibilidad de servicios:** Spring permite el uso de cualquier servicio usando un archivo XML de configuración.
- **Testeo de componentes:** en Spring todos los componentes pueden ser testeados fuera del contenedor.[41]

■ Librerías y Bibliotecas

- **Hibernate (JPA):** se usa para la persistencia de datos y el mapeo objeto-relacional (ORM). Se han usado las anotaciones @Entity, @Table, @Column, @EmbeddedId y @Lob.
- **Lombok:** simplifica la creación de getters y setters, entre otros métodos automáticos. Se han usado las anotaciones @Getter y @Setter.
- **Jackson:** es una librería para trabajar con JSON en Java, se ha usado la anotación @JsonProperty.
- **Java Validation API:** proporciona anotaciones como @Min o @NotBlank para validar las propiedades de los objetos.
- **Java Standard Library:** se ha usado para clases estándar de Java como LocalDate y StandardCharsets.

Aplicaciones Empresariales

Para nuestro trabajo, dotaremos a nuestra aplicación web de una arquitectura empresarial. Estas aplicaciones con una arquitectura orientada a servicios, destacan por estar distribuidas en entornos de intranet o Internet, con servidores backend separados de los clientes frontend. Ofrecen servicios a una variedad de tipos de clientes y otras aplicaciones empresariales. Son escalables y modulares, y pueden soportar muchos usuarios y grandes cargas de trabajo. Estas características permiten una integración flexible, eficiente y segura, que puede crecer con las necesidades de la empresa y adaptarse a cambios sin reestructuraciones completas, lo que garantiza un rendimiento óptimo y una administración simplificada.

Podemos clasificar su arquitectura en las capas que aparecen en la Figura 4.31:

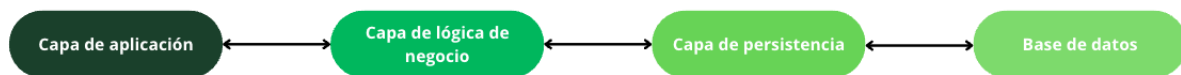


Figura 4.31: Capas de la Arquitectura de las Aplicaciones Empresariales - Elaboración Propia

- **Capa de Aplicación:** es lo que se conoce como Frontend o Interfaz de Usuario, nos centraremos en ella en la siguiente iteración.
- **Capa de Lógica de Negocio:** sirve para implementar las reglas reales del negocio, como cómo se crea, procesa y almacena la información a través de una serie de objetos de negocio o de dominio. Se implementará junto con las siguientes capas en esta iteración.
- **Capa de Servicios:** nueva capa que añadimos, hace accesible a las aplicaciones la lógica de negocio mediante protocolos universales, nosotros usaremos el protocolo REST.
- **Capa Persistencia:** esta capa se define para simplificar el acceso a la base de datos eliminando características específicas de cada SGBD y utilizando lenguajes de consulta como SQL.
- **Base de Datos:** se ha definido previamente en la iteración anterior.

En esta iteración, por tanto, vamos a modelar el Backend de la aplicación, este es la parte del sistema que maneja la lógica, el almacenamiento de datos y la comunicación del servidor, gestionando las operaciones y respuestas a las solicitudes del frontend.

Domain Driven Designment (DDD)

El diseño orientado al dominio (Domain-Driven Design, DDD) enfatiza la importancia de modelar correctamente la capa de dominio. Fomenta la colaboración constante entre desarrolladores y expertos en el dominio para establecer un lenguaje común. Promueve la refactorización continua para capturar mejor los detalles del sistema. DDD se centra en resolver problemas de dominios complejos, destacando la lógica del negocio sobre la parte técnica. Su objetivo es crear aplicaciones comprensibles y eficaces, alineadas con las necesidades empresariales.

Segun el DDD el dominio debe modelarse usando los siguientes elementos:

- **Entidades:** objetos que poseen una identidad distinta y única que los diferencia del resto y se mantiene constante a lo largo del tiempo. Un ejemplo podría ser Finca u Olivo.
- **Objetos-valor:** objetos que no poseen identidad propia, es decir, que no es necesario mantener y se definen completamente por sus atributos. Son inmutables y se usan para describir aspectos de las entidades, como una dirección o una fecha.
- **Servicios:** operaciones o procesos del dominio que no encajan naturalmente en una entidad o un objeto-valor. Proveen funcionalidades específicas sin mantener estado propio. Un ejemplo puede ser un servicio que asigne distintas zonas y fincas.
- **Repositorios:** abstracciones que proporcionan mecanismos para almacenar y recuperar entidades del dominio. Actúan como una colección de entidades, ocultando los detalles de persistencia y suele existir una única instancia. Por ejemplo, RepositorioHistoricoFinca, que permitiría guardar nuevos históricos, eliminarlos o editarlos.
- **Factorías:** se trata de componentes encargados de la creación compleja de objetos del dominio, encapsulando la lógica necesaria para instanciar entidades y agregados de manera adecuada. Por ejemplo, una factoría para gestionar diferentes métodos de pago.
- **Agregados:** conjuntos de entidades y objetos-valor relacionados que se tratan como una única unidad de modificación. Cada agregado tiene una raíz, la cual garantiza la consistencia del conjunto. Por ejemplo, un Ticket y cada línea de él.

Arquitectura de la Aplicación

En este apartado se va a describir la arquitectura de la aplicación, en nuestro caso tendríamos el cliente (Front-end), el servidor (Back-end) y la base de datos, por tanto, podríamos decir que la arquitectura es Cliente-Servidor. Además, la comunicación se realizará mediante una API Restful. En la Figura 4.32 podemos ver el esquema de la arquitectura de la aplicación con algunas de las tecnologías usadas.

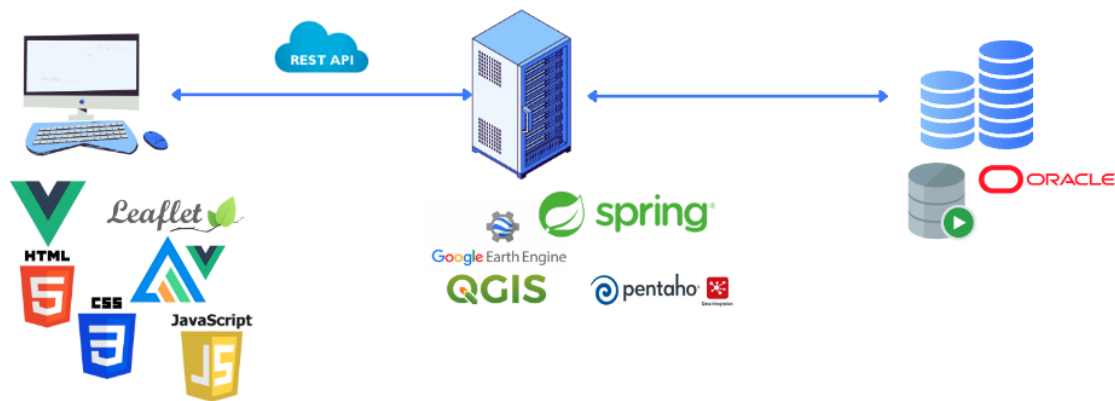


Figura 4.32: Arquitectura de la Aplicación - Elaboración Propia

- **Cliente (Front-end):** implementado mediante Vue JS, es la capa de presentación donde los usuarios interactúan con la aplicación. Maneja el enrutamiento de las vistas y la lógica de la interfaz de usuario.
- **Servidor (Back-end):** se trata del servidor, este maneja la lógica de negocio y se comunica con la base de datos. Aquí, incluimos controladores que reciben solicitudes HTTP desde el cliente, servicios que contienen la lógica de negocio, y repositorios que interactúan con la base de datos.
- **Base de Datos:** implementada en Oracle Database, es donde se almacenan todos los datos persistentes de la aplicación. Spring Boot usa JPA/Hibernate para interactuar con ella.
- **Comunicación (API RESTful):** la comunicación entre el cliente y el servidor se realiza a través de una API RESTful. El cliente hace peticiones HTTP al servidor, que responde con datos en formato JSON.

4.5.2. Capa de Lógica de Negocio

Nuestra capa de dominio, o lógica de negocio, se implementa usando objetos de dominio u objetos de negocio. Estos objetos representan la estructura y funcionamiento de la organización, ofreciendo servicios vitales a la capa de aplicación. Mediante estos servicios, se administran los procesos y reglas operativas fundamentales, garantizando un rendimiento coherente y eficiente.

Esta se va a basar en el diagrama del dominio, explicado en el capítulo 3 y podemos ver en la Figura 3.2.

Para implementar esta capa, declararemos como clases Java cada una de las clases que aparecen en el diagrama, por temas de simplificación, se va a explicar una única clase que será la más completa de todas, esta será HistóricoFinca 4.12.

```
1 public class Historico_finca
2 {
3     @NotNull
4     private LocalDate fecha; //Fecha de la toma de datos
5     @NotNull
6     @Min(0)
7     private Integer municipioCodigo; //Código del municipio
8     @NotNull
9     @Min(0)
10    private Integer provinciaCodigo; //Código de la provincia
11    @NotNull
12    @Min(0)
13    private Integer poligono; //Poligono del código SIGPAC
14    @NotNull
15    @Min(0)
16    private Integer parcela; //Parcela del código SIGPAC
17    @NotNull
18    @Min(0)
19    private Integer recinto; //Recinto del código SIGPAC
20    @JsonProperty("Reflectancia")
21    @NotBlank
22    private String reflectancia; //Índices de vegetación en ese momento
23    @JsonProperty("Temperatura")
24    @NotBlank
25    private String temperatura; //Temperatura en ese momento
26    @NotNull
27    @Min(0)
28    private Double lluvia; //Suma de la lluvia en la fecha de toma de datos
29    @NotBlank
30    private String nombreFuente; //Nombre de la fuente de procedencia
31    @NotBlank
32    private String tipoFuente; //Tipo de la fuente de procedencia
33    //Métodos Getters Setters y constructores
34 }
```

Listado 4.12: Clase HistoricoFinca Capa Dominio

Para implementar esta clase, se han declarado todos sus atributos y métodos, además, para evitar código reiterativo y de escaso interés, se ha usado la validación de beans. Esta se centra en garantizar que los argumentos de operaciones o los atributos de las entidades tengan valores correctos, evitando por tanto inconsistencias.

Podemos ver la validación, por ejemplo, en el atributo lluvia, donde pone *@NotNull* y *@Min(0)* indica que no puede ser nulo y el valor mínimo es 0, para evitar valores negativos. También se ha usado *@JsonProperty* para indicar que ese atributo debe

ser un json o *@NotBlank* para que no esté blanco.

4.5.3. Capa de Persistencia

En esta capa se explicará todo lo necesario para que nuestra aplicación tenga acceso completo a la base de datos.

Conexión con Oracle Database

Para poder conectar nuestra aplicación con la Base de Datos, en primer lugar, a la derecha de IntelliJ aparece un icono de base de datos, debemos crear una nueva conexión de base de datos Oracle como se puede ver en la Figura 4.33. Introducimos nuestros datos y la conexión estaría establecida.

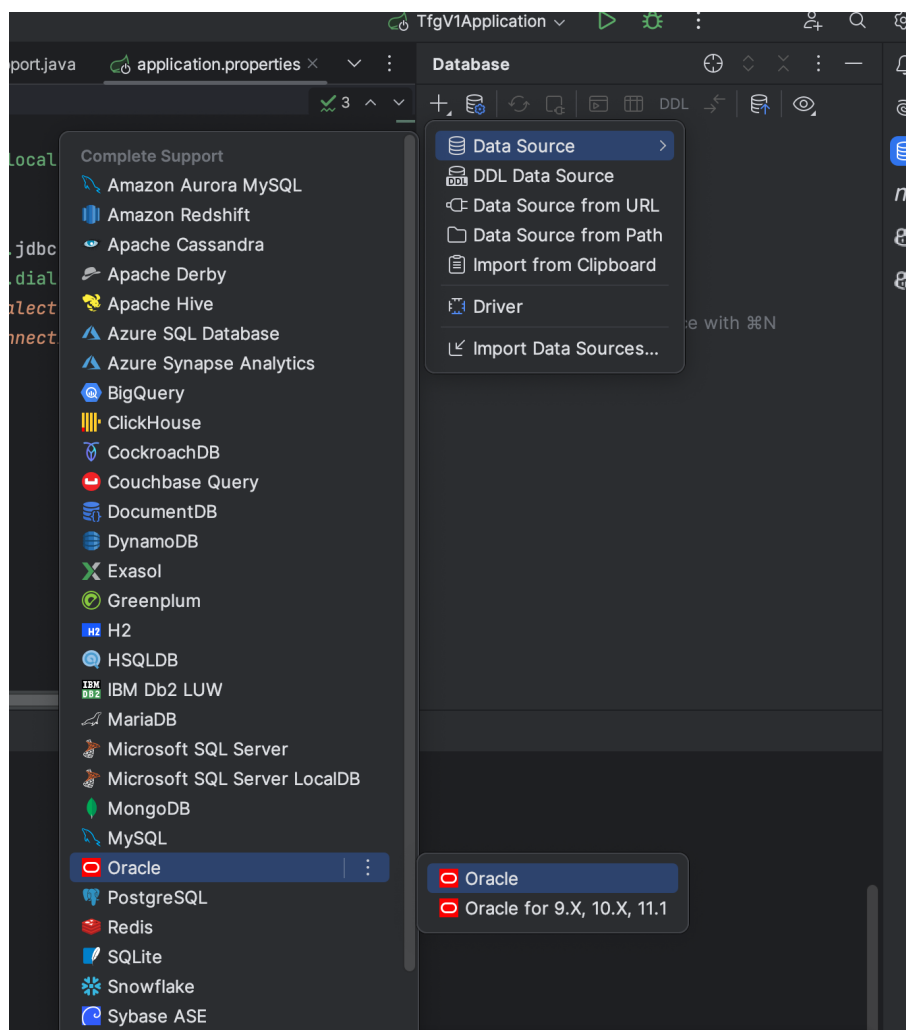


Figura 4.33: Crear Conexión Base de Datos - Elaboración Propia

Para conectar nuestra aplicación a la base de datos previamente enlazada, en el archivo `application.properties` (Listado 4.13), debemos declarar la url, username y password como se ve en el código siguiente. Además de añadir el Driver de Oracle, que también tenemos que añadir la dependencia en el archivo `pom.xml` (Listado 4.14).

```

1 spring.application.name=TFG-V1
2 server.port=4000
3 spring.datasource.url=jdbc:oracle:thin:@//localhost:1521/ORCL
4 spring.datasource.username=xxxxx
5 spring.datasource.password=xxxxx
6 spring.datasource.driver-class-name=oracle.jdbc.OracleDriver
7 spring.jpa.database-platform=org.hibernate.dialect.OracleDialect
8 spring.jpa.properties.hibernate.spatial.dialect=org.hibernate.spatial.dialect.oracle.
   OracleSpatial10gDialect
9 spring.jpa.properties.hibernate.spatial.connection_finder=com.tfg.tfgv1.utils.
   CustomConnectionFinderForSpatialSupport
10 spring.jpa.show-sql=true
11 spring.jpa.hibernate.ddl-auto=update

```

Listado 4.13: `application.properties`

```

1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
   XMLElement-instance"
2     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/
   xsd/maven-4.0.0.xsd">

```

Listado 4.14: Declaración Drive Oracle en `pom.xml`

Adaptación de Clases

Una vez tenemos declaradas todas las clases, debemos convertirlas en entidades, como antes se ha explicado la clase Histórico Finca, se va a seguir con la misma para que se vean bien los campos. El código se puede ver en los Listados 4.15 y 4.16.

```

1 @Entity
2 @Table(name = "HISTORICO_FINCA", schema="ALBAGOMEZ")
3 public class HistoricoFinca
4 {
5     @EmbeddedId
6     private HistoricoFincaId id;
7
8     @JsonProperty("Reflectancia")
9     @Setter @Getter
10    @Column(name = "REFLECTANCIA")
11    @Lob @NotBlank
12    private byte[] reflectancia; //Índices de vegetación en ese momento
13
14    @JsonProperty("Temperatura")
15    @Column(name = "TEMPERATURA")
16    @Setter @Getter
17    @Lob @NotBlank

```

```

18 private byte[] temperatura; //Temperatura en ese momento
19
20 @Min(0)
21 @Setter
22 @Getter
23 private Double lluvia; //Suma de la lluvia en la fecha de toma de datos
24
25 @NotBlank
26 @Column(name = "NOMBRE_FUENTE")
27 @Setter
28 @Getter
29 private String nombreFuente; //Nombre de la fuente de procedencia
30
31 @NotBlank
32 @Column(name = "TIPO_FUENTE")
33 @Setter
34 @Getter
35 private String tipoFuente; //Tipo de la fuente de procedencia
36
37 //Métodos Getters Setters y constructores
38 }

```

Listado 4.15: Clase Historico Finca Capa Persistencia

```

1 @Embeddable
2 public class HistoricoFincaId implements Serializable
3 {
4     @NotNull
5     @Column(name = "FECHA")
6     private LocalDate fecha; //Fecha de la toma de datos
7     @ManyToOne
8     @JoinColumns({
9         @JoinColumn(name = "MUNICIPIO_CODIGO", referencedColumnName = "
10             MUNICIPIO_CODIGO"),
11         @JoinColumn(name = "PROVINCIA_CODIGO", referencedColumnName = "
12             PROVINCIA_CODIGO"),
13         @JoinColumn(name = "POLIGONO", referencedColumnName = "POLIGONO"),
14         @JoinColumn(name = "PARCELA", referencedColumnName = "PARCELA"),
15         @JoinColumn(name = "RECINTO", referencedColumnName = "RECINTO"),
16         @JoinColumn(name = "ZONA_UBICACION", referencedColumnName = "ZONA_UBICACION")
17     })
18     private Finca finca; //Finca
19 ...
20 }

```

Listado 4.16: ID de la Clase Historico Finca Capa Persistencia

Según el DDD 4.5.1, debemos declarar a las clases que necesitan persistencia como entidades, para ello se ha usado la anotación *@Entity*, además, se ha indicado con la anotación *@Table* el nombre de la tabla a la que se refiere de la BD y al esquema de este.

En cuanto a la clave principal de la tabla, vemos que es compuesta, por tanto, tenemos que crear un ID embebido como se ve en el Código de la clase *HistoricoFincaId*,

debe implementar *Serializable* y usar la anotación *@Embeddable*. En ella declaramos la fecha, indicando mediante *@Column* el nombre que tiene en la BD y si nos fijamos, hay muchos atributos que son de la clase Finca, para ello, declaramos la relación con la anotación *@ManyToOne* y mediante *@JoinColumns* indicamos las columnas que necesitamos.

Si seguimos analizando la clase, tenemos el atributo reflectancia y temperatura, la anotación *@JsonProperty()* especifica el nombre del campo cuando se serializa/deserializa en JSON. Las anotaciones *@Setter* y *@Getter* generan automáticamente los métodos setter y getter para la propiedad. *@Column(name =)* indica al campo que corresponde en la base de datos. *@Lob* marca el campo como un tipo de datos grande, adecuado para almacenar grandes objetos binarios. Las demás anotaciones son prácticamente iguales.

También debemos analizar la clase Olivo pues tiene atributos espaciales. Podemos ver su código en el Listado 4.17:

```

1  @Entity
2  @Table(name = "OBJETO", schema="ALBAGOMEZ")
3  @SequenceGenerator(name="ID_GENERATOR", sequenceName="MY_SEQUENCE", initialValue=2663)
4  public class Objeto
5  {
6      @NotNull
7      @Min(-1)
8      @Id
9      @Column(name = "ID_OBJETO")
10     @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "ID_GENERATOR")
11     private Integer idObjeto; //ID del objeto
12
13     @Getter
14     @Column(name="TIPO_OBJETO")
15     private String tipoObjeto; //Tipo de objeto
16
17     @ManyToOne
18     @JoinColumns({
19         @JoinColumn(name = "ZONA_MUNICIPIO_CODIGO", referencedColumnName = "
20             MUNICIPIO_CODIGO"),
21         @JoinColumn(name = "ZONA_PROVINCIA_CODIGO", referencedColumnName = "
22             PROVINCIA_CODIGO"),
23         @JoinColumn(name = "ZONA_UBICACION", referencedColumnName = "UBICACION")
24     })
25     private Zona zona;
26     @Column(name="POLIGONO_ENVOLVENTE",columnDefinition = "MDSYS.SDO_GEOMETRY")
27     private Polygon poligonoEnvolvente;
28
29     @Column(name="PUNTO_MEDIO",columnDefinition = "MDSYS.SDO_GEOMETRY")
30     private Point puntoMedio;
31     ...
32 }

```

Listado 4.17: Clase Objeto Capa Persistencia

De esta clase tenemos que destacar, en primer lugar, las anotaciones `@SequenceGenerator` y `@GeneratedValue`, que se usan en JPA para configurar la generación automática de identificadores primarios. `@SequenceGenerator` define un generador de secuencia, donde `name="ID_GENERATOR"` es el nombre del generador, `sequenceName="MY_SEQUENCE"` es el nombre de la secuencia en la base de datos, e `initialValue=2663` establece el valor inicial de la secuencia. `@GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "ID_GENERATOR")` utiliza este generador específico para asignar valores de identificador a nuevas entidades, asegurando que los identificadores se incrementen de acuerdo a la secuencia definida.

Por otro lado, las anotaciones `@Column(name="POLIGONO_ENVOLVENTE", columnDefinition = "MDSYS.SDO_GEOMETRY")` y `@Column(name="PUNTO_MEDIO", columnDefinition = "MDSYS.SDO_GEOMETRY")` se utilizan para mapear los atributos `poligonoEnvolvente` y `puntoMedio` a las columnas `POLIGONO_ENVOLVENTE` y `PUNTO_MEDIO` en la base de datos, respectivamente. Ambas columnas están definidas con el tipo de datos espacial `MDSYS.SDO_GEOMETRY` de Oracle. El atributo `poligonoEnvolvente` es de tipo `Polygon`, representando un polígono, y `puntoMedio` es de tipo `Point`, representando un punto.

Creación de Repositorios

El siguiente paso sería la creación de los repositorios para acceder a los datos de la BD. Implementar repositorios en Spring es fundamental porque permite una clara separación de responsabilidades y simplifica la gestión del código. Al usar repositorios, se aísla la lógica de acceso a datos de la lógica de negocio, facilitando el mantenimiento y la evolución de la aplicación.

Spring Data JPA permite definir repositorios mediante interfaces, lo que reduce significativamente el código repetitivo necesario para operaciones comunes como Crear, Leer, Actualizar y Borrar (CRUD). Además, las consultas pueden definirse mediante convenciones de nombres, y Spring se encarga de generar automáticamente las consultas SQL necesarias, eliminando la necesidad de escribirlas manualmente.

A continuación voy a describir el Repositorio más completo de la aplicación, *HistoricoFincaRepositorio*, este se encuentra en el Listado 4.18:

```
1 @Repository
2 @Transactional(propagation = Propagation.REQUIRED)
3 public class HistoricoFincaRepositorio
4 {
5     @PersistenceContext
```

```

6 | EntityManager em;
7 |
8 | @Transactional(propagation = Propagation.SUPPORTS, readOnly = true)
9 | public Optional<HistoricoFinca> buscar(HistoricoFincaId id)
10 | {
11 |     return Optional.ofNullable(em.find(HistoricoFinca.class, id));
12 | }
13 |
14 | public void guardar(HistoricoFinca historicoFinca)
15 | {
16 |     em.persist(historicoFinca);
17 | }
18 |
19 | public void actualizar(HistoricoFinca historicoFinca)
20 | {
21 |     em.merge(historicoFinca);
22 | }
23 |
24 | public void borrar(HistoricoFinca historicoFinca)
25 | {
26 |     em.remove(em.merge(historicoFinca));
27 | }
28 |
29 | /**
30 |  * @brief Obtener las Fincas que tienen datos históricos
31 |  * @return Lista de Fincas
32 |  */
33 | @Transactional(propagation = Propagation.SUPPORTS, readOnly = true)
34 | public List<Finca> obtenerFincasConHistorico()
35 | {
36 |     String jpql = "SELECT DISTINCT hf.id.finca FROM HistoricoFinca hf";
37 |     TypedQuery<Finca> query = em.createQuery(jpql, Finca.class);
38 |     return query.getResultList();
39 | }
40 | ...
41 | }
42 | }

```

Listado 4.18: Repositorio clase Historico Finca

En primer lugar, la etiqueta *@Repository* en Spring se utiliza para indicar que una clase es un componente de acceso a datos (DAO) y también proporciona la capacidad de traducir automáticamente excepciones específicas de la base de datos en excepciones de Spring.

La anotación *@Transactional(propagation = Propagation.REQUIRED)* se utiliza para gestionar transacciones, asegurando que los métodos dentro de la clase se ejecuten dentro de una transacción. El *propagation = Propagation.REQUIRED* indica que si ya existe una transacción en curso, el método se ejecutará dentro de esa transacción; si no, se creará una nueva transacción.

Por otro lado, la anotación *@PersistenceContext* se utiliza para inyectar una instancia de *EntityManager*, que es la interfaz principal para interactuar con la base de

datos en JPA (Java Persistence API). En este caso, *EntityManager* se inyecta para realizar operaciones de persistencia.

En cuanto a los métodos, el método buscar está anotado con *@Transactional(propagation = Propagation.SUPPORTS, readOnly = true)*, lo que indica que este método es de solo lectura y no requiere una nueva transacción. Si existe una transacción en curso, el método se ejecutará dentro de esa transacción. Se han implementado los cuatro métodos básicos en cada repositorio, buscar, guardar y actualizar. Ya dependiendo de lo que el Frontend nos requiriera, se implementaron funciones nuevas como *obtenerFincasConHistorico()* en la que ya debíamos hacer una sentencia SQL más compleja para acceder a ella.

Creación del SistemaFincas

Una vez modificado las clases e implementado los repositorios, podríamos acceder sin problema a la BD, sólo nos quedaría implementar un sistema. En nuestro caso, se corresponde con la clase *SistemaFincas*, que será el encargado de realizar todas las operaciones necesarias en nuestra aplicación web. El código de este es muy extenso ya que soporta todas las operaciones así que se va a incluir la definición de la clase en el Listado 4.21 y algunos métodos más significativos.

```
1 @Service
2 @Validated
3 public class SistemaFincas
4 {
5     @Autowired
6     ProvinciaRepositorio provinciaRepositorio;
7     @Autowired
8     MunicipioRepositorio municipioRepositorio;
9     @Autowired
10    FincaRepositorio fincaRepositorio;
11    ...
}
```

Listado 4.19: Declaración SistemaFincas

En el código 4.21, se ha declarado el sistema con las anotaciones *@Service* que indica que es un servicio, es decir, un componente de la capa de servicio que contiene la lógica de negocio y *@Validated* que se usa para activar automáticamente la validación de métodos en la clase.

Por otro lado, ya dentro de la clase, las anotaciones *@Autowired* se utilizan para inyectar automáticamente las dependencias de los repositorios, permitiendo que el servicio utilice estos repositorios para realizar operaciones de acceso a datos, como

búsquedas y persistencia, sin necesidad de inicializarlos manualmente, facilitando la gestión de dependencias y promoviendo la inversión de control (IoC).

```

1  ...
2  @Transactional
3  public List<Cosecha> obtenerDatosCosechas(int provinciaCodigo, int municipioCodigo,
4                                          int poligono, int parcela, int recinto )
5  {
6      Optional<Finca> finca = fincaRepositorio.buscarFinca(provinciaCodigo,
7                                                         municipioCodigo, poligono, parcela, recinto);
8      if(finca.isEmpty())
9          throw new IllegalArgumentException("Finca no encontrada");
10     else
11     {
12         return cosechaRepositorio.obtenerCosechasPorFinca(finca.get());
13     }
14 }
15
16 @Transactional
17 public Optional<Finca> obtenerFincaOliveo(int oliveoId)
18 {
19     return objetoRepositorio.obtenerFincaObjeto(oliveoId);
20 }
21 ...

```

Listado 4.20: Ejemplo métodos ServicioFinca

En estos métodos seleccionados en el Listado 4.20, vemos la anotación *@Transactional* que garantiza que el código dentro del método anotado se ejecute dentro de una transacción. Siempre va a devolver *Optional* por si la Base de Datos no contiene el objeto a buscar y se llama a las funciones de los repositorios.

Pruebas

Para cada uno de los métodos del sistema, se ha realizado un test en la clase *SistemaFincasTest* para verificar su funcionamiento y prevenir errores que puedan suceder en otras etapas de la implementación del sistema, se ha usado JUnit y Spring Test Framework. De nuevo, el código es muy extenso así que se presentarán algunos de ellos en el Listado 4.21.

```

1  @ExtendWith(SpringExtension.class)
2  @SpringBootTest(classes = TfgV1Application.class)
3  @Transactional
4  public class SistemaFincasTest
5  {
6      @Autowired
7      SistemaFincas sistemaFincas;
8
9      @Test
10     public void accesoSistema()

```

```

11 {
12     Assertions.assertThat(sistemaFincas).isNotNull();
13 }
14
15 @Test
16 @DirtiesContext(methodMode = DirtiesContext.MethodMode.AFTER_METHOD)
17 public void testBuscarProvincia()
18 {
19     Integer codigoProvincia = 23;
20     Provincia provincia = new Provincia(23, "Jaén");
21     Optional<Provincia> optional=sistemaFincas.buscarProvincia(codigoProvincia);
22     Assertions.assertThat(optional.isPresent()).isTrue();
23     Assertions.assertThat(optional.get().getCodigoProvincia()).isEqualTo(provincia.
        getCodigoProvincia());
24     Assertions.assertThat(optional.get().getNombreProvincia()).isEqualTo(provincia.
        getNombreProvincia());
25 }
26 ...
27 @Test
28 @DirtiesContext(methodMode = DirtiesContext.MethodMode.AFTER_METHOD)
29 public void buscarObjetosPorZonaConHistorico()
30 {
31     int provinciaCodigo=23;
32     int municipioCodigo=5;
33     String zonaUbicacion="J3";
34
35     Optional<Provincia> optional=sistemaFincas.buscarProvincia(provinciaCodigo);
36     MunicipioId municipioId = new MunicipioId(municipioCodigo, optional.get());
37     Optional<Municipio> optionalMunicipio = sistemaFincas.buscarMunicipio(municipioId
        );
38     ZonaId zonaId = new ZonaId(zonaUbicacion, optionalMunicipio.get());
39     Optional<Zona> optionalZona = sistemaFincas.buscarZona(zonaId);
40     List<Objeto> resultado = sistemaFincas.obtenerObjetosZonaConHistorico(
        optionalZona.get());
41     Assertions.assertThat(resultado.size()).isGreaterThan(0);
42 }
43 @Test
44 @DirtiesContext(methodMode = DirtiesContext.MethodMode.AFTER_METHOD)
45 public void buscarHistoricoFincaPorAnio()
46 {
47     int provinciaCodigo=23;
48     int municipioCodigo=32;
49     int poligono=3;
50     int parcela=101;
51     int recinto=3;
52     int anio=2023;
53
54     Optional<Finca> resultado=sistemaFincas.buscarFincaEsp(provinciaCodigo,
        municipioCodigo, poligono, parcela, recinto);
55     if(resultado.isPresent())
56     {
57         List<HistoricoFinca> historico = sistemaFincas.obtenerHistoricoFincaAnioSat(
            resultado.get(), anio);
58         Assertions.assertThat(historico.size()).isGreaterThan(0);
59     }
60 }
61 ...

```

Listado 4.21: Declaración SistemaFincas

En primer lugar, se han usado las anotaciones `@ExtendWith` para integrar el soporte de Spring en JUnit 5, `@SpringBootTest` que indica que esta es una prueba de Spring Boot que cargará el contexto de la aplicación `TfgV1Application`. Esto permite que la prueba se ejecute con todas las configuraciones y beans definidos en la aplicación. Además de `@Transactional` explicada anteriormente.

`@autowired` inyecta una instancia del servicio `SistemaFincas` en la clase de prueba. El primer test siempre es para verificar que el acceso al sistema sea correcto.

En cuanto a las pruebas, `@DirtiesContext(methodMode = DirtiesContext.MethodMode.AFTER_METHOD)` indica que el contexto de Spring se debe limpiar después de la ejecución del método de prueba, asegurando que no haya efectos secundarios entre las pruebas. Ya es mediante Assertions donde vamos viendo, para cada ejecución, qué resultado esperamos.

4.5.4. Capa de Servicios

El último paso en el desarrollo del backend es la creación de un servicio REST (Representational State Transfer). Un servicio RESTful es crucial para permitir que diferentes clientes interactúen con el servidor de manera sencilla y eficiente. Al implementar un servicio REST, se garantiza una interfaz estandarizada para la comunicación entre el frontend y el backend, facilitando operaciones como la creación, lectura, actualización y eliminación de datos a través de endpoints HTTP.

Creación de DTOs

En primer lugar, hay que crear los DTOs de cada uno de los objetos que intervengan en las peticiones REST. Los DTOs (Data Transfer Objects) son clases que se utilizan para transportar datos entre capas de una aplicación, especialmente entre la capa de presentación y la capa de servicios. Su propósito principal es encapsular los datos necesarios para una operación específica y evitar el acoplamiento directo entre las capas de la aplicación.

Se ha creado un DTO para cada una de las clases que intervienen, a continuación se muestra el de Histórico Finca en el Listado [4.22](#).

```
1 public record DTOHistoricoFinca(  
2     LocalDate fecha,  
3     Integer fincaMunicipioCodigo,
```

```

4      Integer fincaProvinciaCodigo,
5      Integer fincaPoligono,
6      Integer fincaParcela,
7      Integer fincaRecinto,
8      String zonaUbicacion,
9      byte[] reflectancia,
10     byte[] temperatura,
11     Double lluvia,
12     String nombreFuente,
13     String tipoFuente)
14 {
15     public DTOHistoricoFinca(HistoricoFinca historicoFinca)
16     {
17         this(
18             historicoFinca.getFecha(),
19             historicoFinca.getMunicipioCodigo(),
20             historicoFinca.getProvinciaCodigo(),
21             historicoFinca.getPoligono(),
22             historicoFinca.getParcela(),
23             historicoFinca.getRecinto(),
24             historicoFinca.getZonaUbicacion(),
25             historicoFinca.getReflectancia(),
26             historicoFinca.getTemperatura(),
27             historicoFinca.getLluvia(),
28             historicoFinca.getNombreFuente(),
29             historicoFinca.getTipoFuente()
30         );
31     }
32 }

```

Listado 4.22: DTOHistoricoFinca

Creación de Controlador REST

A continuación, se ha creado el controlador REST, como podemos ver en el Código 4.23, la anotación *@RestController* utiliza para marcar una clase como un controlador REST, lo que significa que sus métodos son automáticamente asociados con las solicitudes HTTP y devuelven objetos serializados como respuestas HTTP.

Por otro lado, *@RequestMapping("/TFG")* especifica la ruta base para todas las solicitudes manejadas por este controlador. Además, dentro de la clase ControladorREST, el servicio SistemaFincas se inyecta automáticamente utilizando *@Autowired*.

```

1 @RestController
2 @RequestMapping("/TFG")
3 public class ControladorREST
4 {
5     @Autowired
6     SistemaFincas sistemaFincas;
7
8     /**
9     * Handler para excepciones de violación de restricciones

```

```

10  */
11  @ExceptionHandler({ConstraintViolationException.class})
12  @ResponseStatus(HttpStatus.BAD_REQUEST)
13  public ResponseEntity<String> handlerViolacionRestricciones(
14      ConstraintViolationException e)
15  {
16      return ResponseEntity.badRequest().body(e.getMessage());
17  }

```

Listado 4.23: Controlador REST

En el controlador, se partió agregando peticiones GET y POST para todas las entidades pero no fue hasta el desarrollo de la interfaz de usuario cuando se mejoró y añadieron más peticiones según se fueron necesitando. Como el código es muy extenso, se van a explicar algunos ejemplos en el Listado 4.24:

```

1  ...
2  /**
3   * Consulta de Finca
4   * @param idP id de la provincia
5   * @param idM id del municipio
6   * @param ubZ nombre de la zona
7   * @param poligono poligono de la finca
8   * @param parcela parcela de la finca
9   * @param recinto recinto de la finca
10  */
11  @GetMapping("/provincias/{idP}/municipios/{idM}/zonas/{ubZ}/fincas/{poligono}/{
12      parcela}/{recinto}")
13  ResponseEntity<DTOfinca> verFinca(@PathVariable Integer idP, @PathVariable Integer
14      idM, @PathVariable String ubZ,
15      @PathVariable Integer poligono, @PathVariable
16      Integer parcela, @PathVariable Integer recinto)
17  {
18      Optional<Provincia> p = sistemaFincas.buscarProvincia(idP);
19      if(p.isPresent())
20      {
21          Optional<Municipio> m = sistemaFincas.buscarMunicipio(new MunicipioId(idM, p.
22              get()));
23          if(m.isPresent())
24          {
25              Optional<Zona> z = sistemaFincas.buscarZona(new ZonaId(ubZ, m.get()));
26              if(z.isPresent())
27              {
28                  Optional<Finca> f = sistemaFincas.buscarFinca(new FincaId(poligono,
29                      parcela, recinto, z.get()));
30                  return f
31                      .map(u -> ResponseEntity.ok(new DTOfinca(u)))
32                      .orElse(ResponseEntity.notFound().build());
33              }
34          }
35          else
36              return ResponseEntity.notFound().build();
37      }
38      else
39          return ResponseEntity.notFound().build();
40  }
41  else
42      return ResponseEntity.notFound().build();

```

```

36         return ResponseEntity.notFound().build();
37     }
38     /**
39     * @brief Función para obtener las fincas con histórico
40     * @return Lista de Fincas
41     */
42     @GetMapping("/FincasConHistorico")
43     @ResponseStatus(HttpStatus.OK)
44     public List<DTOFinca> obtenerFincasConHistorico()
45     {
46         return sistemaFincas.obtenerFincasConHistorico()
47             .stream().map(DTOFinca::new).toList();
48     }
49     @GetMapping("/{provincia}/{provinciaCodigo}/municipio/{municipioCodigo}/finca/{poligono}
50         {/parcela}/{recinto}")
51     public ResponseEntity<Finca> buscarFincaEsp(
52         @PathVariable int provinciaCodigo,
53         @PathVariable int municipioCodigo,
54         @PathVariable int poligono,
55         @PathVariable int parcela,
56         @PathVariable int recinto)
57     {
58         Optional<Finca> fincaOptional = sistemaFincas.buscarFincaEsp(provinciaCodigo,
59             municipioCodigo, poligono, parcela, recinto);
60         return fincaOptional.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.
61             notFound().build());
62     }
63     ...

```

Listado 4.24: Controlador REST - Peticiones

1. **verFinca:** se trata de una solicitud *GET* que permite consultar los detalles de una finca específica. Toma varios parámetros de ruta mediante la anotación *@PathVariable*. Internamente, el método busca la provincia, el municipio, la zona y la finca correspondientes utilizando el servicio *sistemaFincas*. Si la finca se encuentra, devuelve una respuesta *ResponseEntity* con el objeto *DTOFinca* correspondiente. Si no se encuentra, devuelve una respuesta de *error 404* (no encontrado).
2. **obtenerFincasConHistorico:** esta solicitud *GET* devuelve una lista de todas las fincas que tienen historial asociado. No toma ningún parámetro de ruta y simplemente devuelve una lista de *DTOFinca*, que son objetos de transferencia de datos que contienen información sobre las fincas con historial.
3. **buscarFincaEsp:** esta solicitud *GET* permite buscar una finca específica por su provincia, municipio, polígono, parcela y recinto. Toma estos parámetros como parte de la ruta y utiliza el servicio *sistemaFincas* para buscar la finca correspondiente. Si se encuentra, devuelve una respuesta *ResponseEntity* con el objeto *Finca* correspondiente. Si no se encuentra, devuelve una respuesta de *error 404* (no encontrado).

4.6. Cuarta iteración: Diseño de la Página Web

4.6.1. Introducción

En esta sección se va a proceder a diseñar la página web, es decir, la interfaz de usuario de la aplicación. En primer lugar, se mostrarán los diferentes Mock Ups de ella y luego la implementación en Vue JS.

El código para esta iteración y la anterior se encuentra disponible en el siguiente repositorio de [GitHub](#).

Estudio de alternativas y viabilidad

A la hora de diseñar la interfaz de usuario fue donde más decisiones se tomaron. En primer lugar, se decidió que el Framework para el Frontend fuera Vue JS ya tiene mucha facilidad de uso, flexibilidad y rendimiento. Es muy intuitivo y tiene una curva de aprendizaje suave, lo que nos permite construir interfaces de usuario dinámicas y complejas con relativa rapidez. Además, su ecosistema es robusto, con una gran cantidad de bibliotecas y herramientas que pueden integrarse fácilmente, como bibliotecas para representar mapas y gráficas, que era lo que más nos interesaba. Como era algo nuevo para aprender, un factor determinante fue la gran documentación que hay al respecto.

Tecnologías usadas

Destacar, al igual que en la iteración anterior [4.3](#), el entorno de desarrollo fue IntelliJ.

■ Lenguajes de Programación

- **HTML5:** debido a que es el lenguaje estándar más utilizado para estructurar y presentar contenido en la web, se ha usado en el desarrollo frontend de la aplicación. Es fundamental para la creación de la estructura básica de las páginas web y soporta una amplia gama de elementos multimedia, gráficos y semánticos.
- **CSS (Cascading Style Sheets):** ha sido utilizado para describir la presentación de la página web. CSS controla el diseño visual, permitiendo aplicar

estilos como colores, fuentes y diseño de la página, asegurando una presentación coherente y atractiva.

- **JavaScript:** es un lenguaje de programación interpretado, principalmente usado para crear contenido web dinámico e interactivo. Permite la manipulación del DOM, la implementación de funcionalidades complejas y la comunicación asíncrona con servidores, mejorando significativamente la experiencia del usuario en las aplicaciones web. Se ha utilizado en el desarrollo frontend ya que Vue JS es un marco JavaScript para crear interfaces de usuario.
- **TypeScript:** en ciertos aspectos del frontend desarrollado con Vue JS, se ha usado TypeScript ya que añade tipado estático mejorando la seguridad, robustez, mantenibilidad y escalabilidad del código, facilitando el desarrollo de aplicaciones web complejas.

■ Frameworks

Vue JS

El Framework elegido para el desarrollo Frontend fue Vue JS. Este, es un marco de JavaScript para crear interfaces de usuario. Se basa en HTML, CSS y JavaScript estándar (Figura 4.34) y proporciona un modelo de programación declarativo basado en componentes que le ayuda a desarrollar eficientemente interfaces de usuario de cualquier complejidad [42].

Vue cubre la mayoría de las características comunes necesarias en el desarrollo frontend. Teniendo en cuenta que el desarrollo web es muy diverso, Vue está diseñado para ser flexible y adaptable de forma incremental. Dependiendo de su caso de uso, Vue se puede utilizar de diferentes maneras:

- Mejora del HTML estático sin un paso de compilación
- Incrustar como componentes web en cualquier página
- Solicitud de una sola página (SPA)
- Fullstack / Representación del lado del servidor (SSR)
- Jamstack / Generación de sitios estáticos (SSG)



Figura 4.34: Vue JS - Lenguaje JS

Ventajas e inconvenientes

Vue JS fue elegido porque tiene una sintaxis clara y una API intuitiva, lo que facilita el aprendizaje y la adopción para desarrolladores de todos los niveles de experiencia. Su enfoque progresivo permite a los desarrolladores utilizar solo las características que necesitan, lo que hace que la experiencia de desarrollo sea más fácil de entender.

Aunque Angular y React son también opciones sólidas para el desarrollo frontend, Vue destaca por:

- **Flexibilidad y escalabilidad:** puede manejar aplicaciones de cualquier tamaño y complejidad. Ofrece opciones avanzadas para la gestión del estado, el enrutamiento o la validación de formularios.
- **Rendimiento:** está hecho para funcionar rápidamente. Para mejorar el rendimiento de la aplicación y reducir las actualizaciones del DOM, utiliza un sistema de virtual DOM.
- **Gran comunidad y ecosistema:** Vue.js cuenta con una comunidad activa y creciente de desarrolladores que contribuyen con bibliotecas, herramientas y recursos educativos.

■ Librerías y Bibliotecas

- **Bootstrap:** se trata de un framework de CSS de código abierto que facilita el diseño de sitios web responsivos y móviles. Ofrece componentes predefinidos y estilos, acelerando el desarrollo y asegurando una apariencia consistente.
- **vue3-apexcharts:** es una biblioteca de gráficos para Vue JS 3 que facilita la creación de visualizaciones de datos interactivas y atractivas. Ofrece una amplia variedad de tipos de gráficos como líneas, barras, áreas, etc.
- **Leaflet:** es una librería JavaScript para generar mapas interactivos. Dentro de este, se ha usado el plugin *leaflet-draw* que permite dibujar y editar

formas directamente en los mapas, añadiendo funcionalidad avanzada de edición geoespacial, *leaflet/dist/leaflet.css* que define la presentación básica de los mapas y *leaflet.markercluster* para gestionar múltiples marcadores.

- **Turf JS:** es una potente biblioteca de JavaScript para el análisis geoespacial y la manipulación de datos geográficos. Dentro de esta, se ha usado el módulo *@turf/boolean-point-in-polygon* que permite verificar si un punto específico se encuentra dentro de un polígono dado.
- **Proj4:** es una biblioteca esencial para trabajar con coordenadas geoespaciales, especializada en la proyección de coordenadas. Se ha usado para convertir coordenadas del sistema UTM a latitud y longitud, permitiendo la integración y manipulación de datos geoespaciales en diferentes formatos y sistemas de referencia.

4.6.2. Diseño de la Interfaz de Usuario

La interfaz de usuario es el principal punto de contacto entre el usuario y el sistema, por ello, esta debe de ser usable y accesible ya que si no, puede producir frustración en el usuario. La interfaz se utiliza para presentar y manejar los datos que se obtienen, por tanto, debe ser independiente a la capa de dominio.

Se han desarrollado estándares para el desarrollo de la interfaz de usuario como [ISO 9241](#) (Ergonomía de la interacción persona-sistema) o [ISO 14915-2002](#) (Ergonomía del software para interfaces de usuario multimedia).

En los sistemas interactivos, el diálogo con el usuario cobra gran importancia, ya que la interfaz determina la percepción e impresión que el usuario posee de la aplicación. Los prototipos permiten incorporar las impresiones del usuario en la interfaz ya que permite probar partes del sistema para verificar funcionalidades. Son una herramienta muy útil para hacer participar activamente al usuario en el desarrollo y poder evaluar el producto desde las primeras fases del desarrollo.

Hay muchas técnicas para la construcción de prototipos, algunas dan más importancia a la presentación y otras a la interacción. Estas son: bocetos (sketching), storyboards, prototipos de papel, maquetas, storyboards navegables, wireframes, escenarios y prototipos de software.

Wireframes

En primer lugar, se va a presentar el prototipo de la interfaz de usuario usando Wireframes. Este término deriva de los esqueletos de alambre usados por escultores y modeladores. Son guías visuales que representan el esqueleto de las interfaces del sistema interactivo. Permiten crear una representación esquemática, sin elementos gráficos ni comportamiento, de una página web o una pantalla de una aplicación, facilitando la comunicación de las decisiones de diseño. Los wireframes se van a realizar en la herramienta Visual Paradigm.

- **Página Principal (Fig 4.35):** Wireframe de la página de inicio, se estructura con un mapa y un buscador de fincas, una tabla con información sobre enfermedades y un panel con información sobre índices de vegetación.

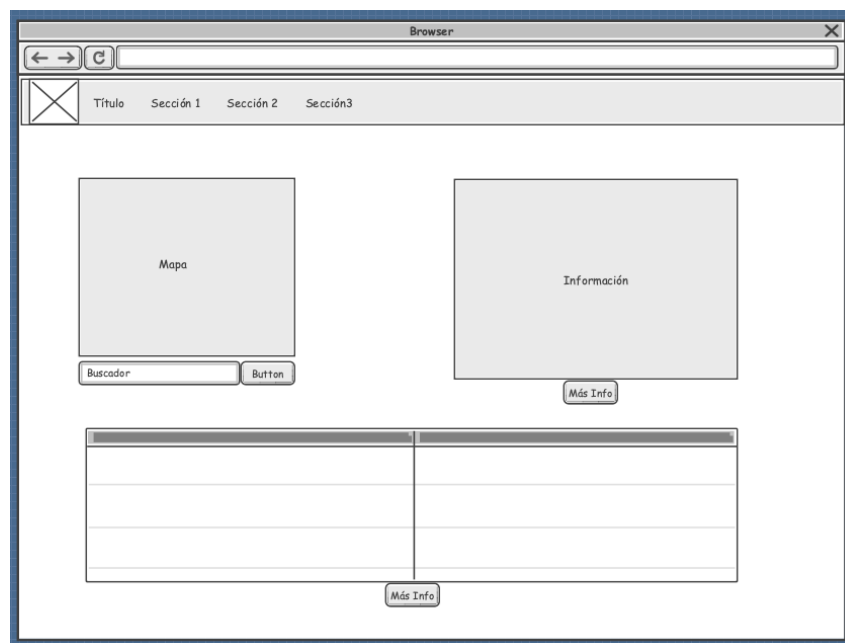


Figura 4.35: Wireframe Página Principal - Elaboración Propia

- **Página Fincas (Fig 4.36):** Wireframe de la página donde se pondrá el buscador de fincas y aparecerán las más completas para mayor rapidez de búsqueda.

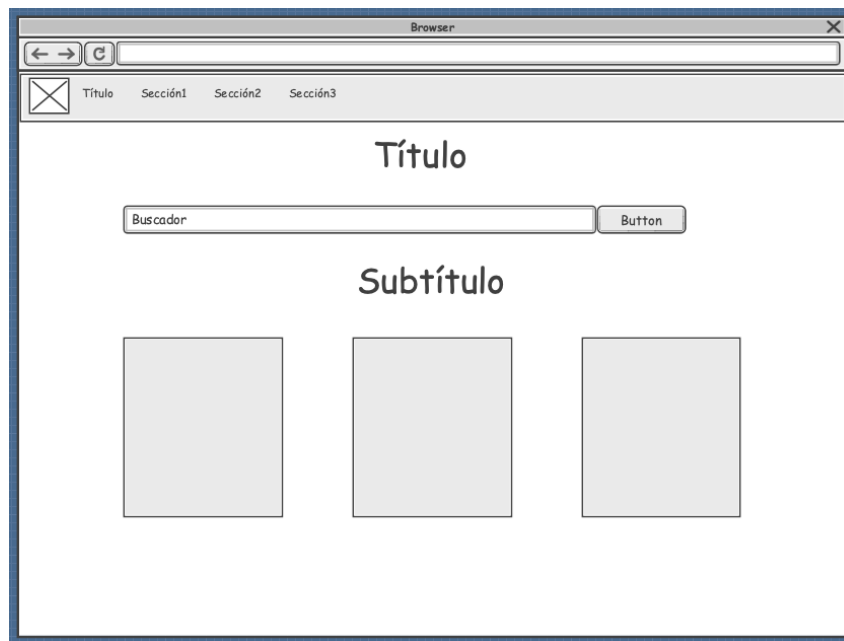


Figura 4.36: Wireframe Página Fincas - Elaboración Propia

- **Página Finca (Fig 4.37):** Wireframe de la página específica de una finca, donde aparecerá toda la información de ella a la izquierda, a la derecha aparecerá un mapa con el contorno de la finca y si hay olivos disponibles y debajo todos los gráficos sobre ella.

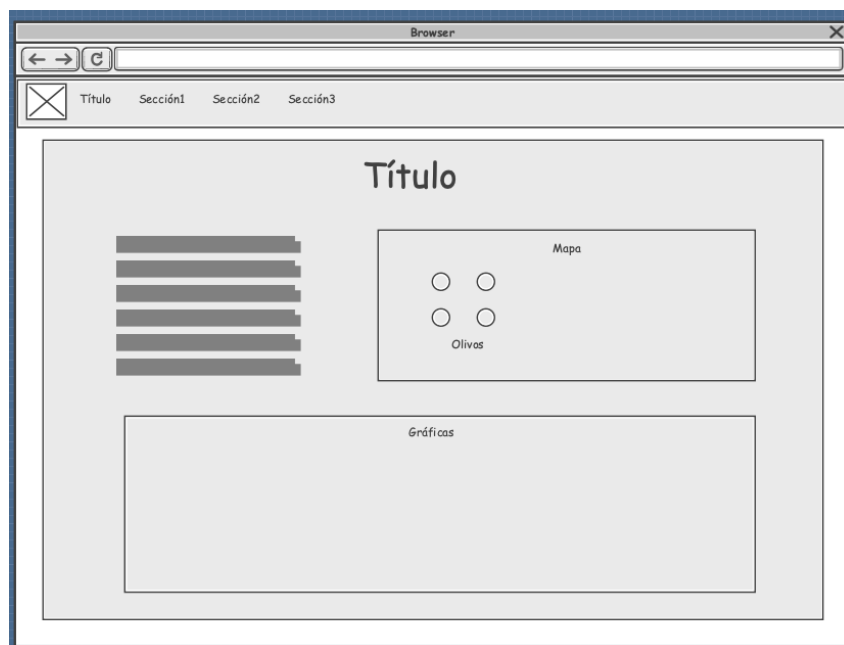


Figura 4.37: Wireframe Página Finca Específica - Elaboración Propia

- **Página Olivo (Fig 4.38):** Wireframe de la página específica de un olivo, aparecerá toda su información y las gráficas específicas de él.

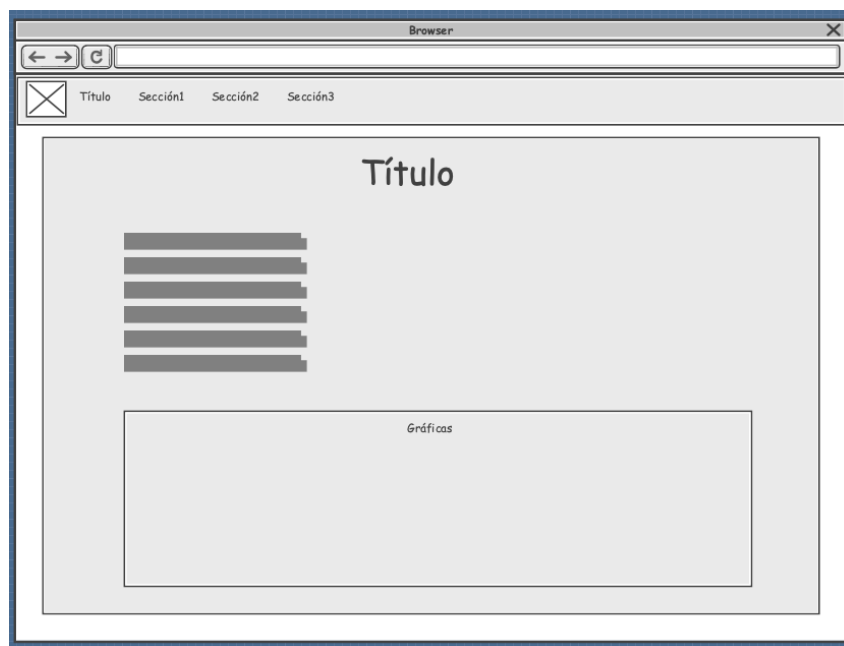


Figura 4.38: Wireframe Página Olivo Específico - Elaboración Propia

- **Página Índices de Vegetación (Fig 4.39):** Wireframe de la página donde se explicarán los diferentes índices de vegetación, por cada panel se explicará uno.

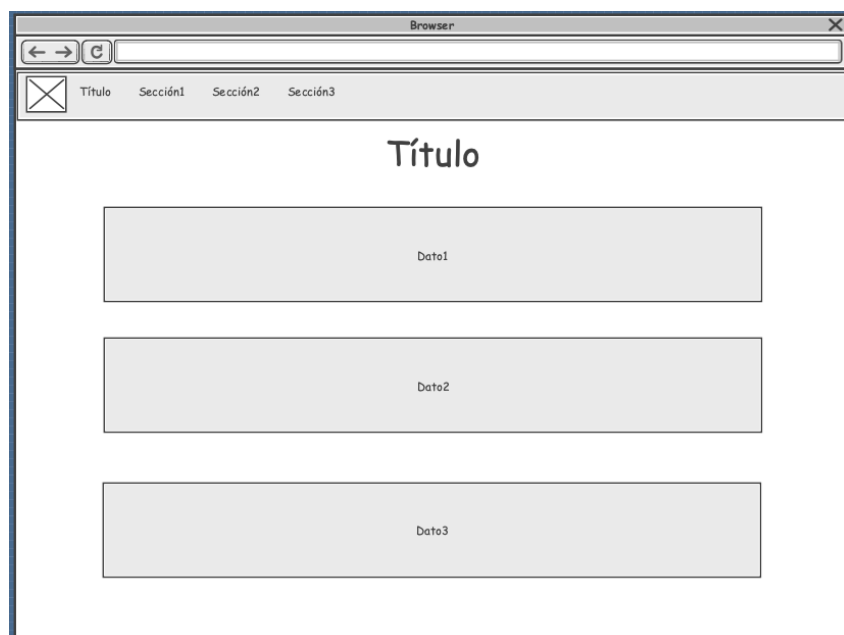


Figura 4.39: Wireframe Página Índices Vegetación - Elaboración Propia

- **Página Tabla Enfermedades (Fig 4.40):** Wireframe de la página donde se explicarán las diferentes enfermedades más comunes del olivar en una tabla indicando el grado de incidencia de los posibles factores.

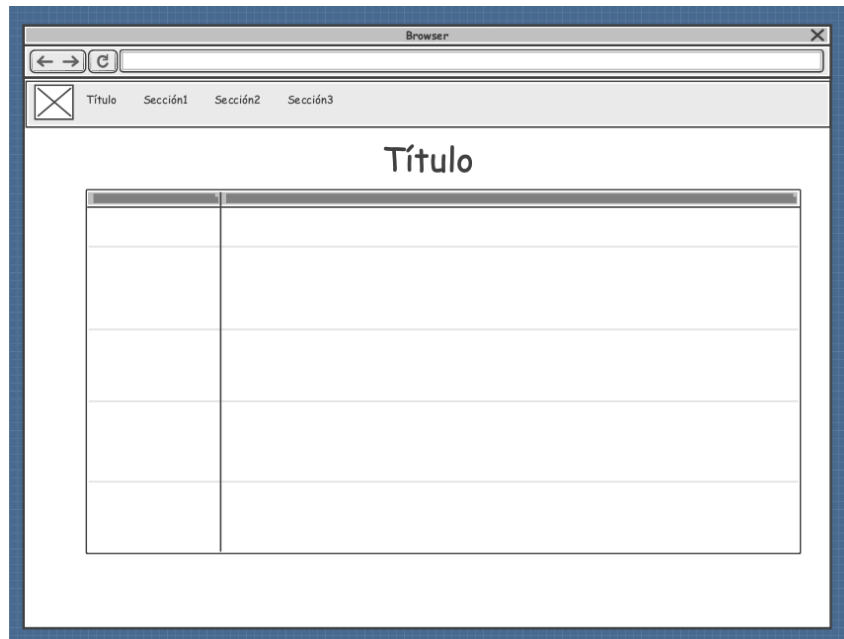


Figura 4.40: Wireframe Página Tabla Enfermedades - Elaboración Propia

Mock-Ups

Una vez explicados los Wireframes, pasaremos a los Mock-Ups. Estos, son representaciones visuales detalladas que muestran el diseño y la estructura final de la interfaz de usuario. Incluyen elementos como colores, tipografías e imágenes junto con la disposición de los componentes en la página.

A diferencia de los wireframes, que son esquemáticos y básicos, los mockups ofrecen una vista más realista de cómo se verá el producto final. Se utilizan para obtener feedback y realizar ajustes antes del desarrollo real. Son una herramienta clave en la fase de diseño para alinear expectativas entre diseñadores, desarrolladores y clientes.

Cabe destacar que los Mock-Ups fueron desarrollados antes de realizar la interfaz de usuario, por ello, puede haber funcionalidades que difieran en lo explicado en los siguientes apartados. Estos, se realizaron con la herramienta [MarvelApp](#).

- **Página Principal (Fig 4.41):** Mock-up de la página de inicio, a la izquierda podemos ver un mapa donde aparecerán todas las fincas de la base de datos junto con un buscador, a la derecha información sobre los índices de vegetación y abajo información sobre las enfermedades.

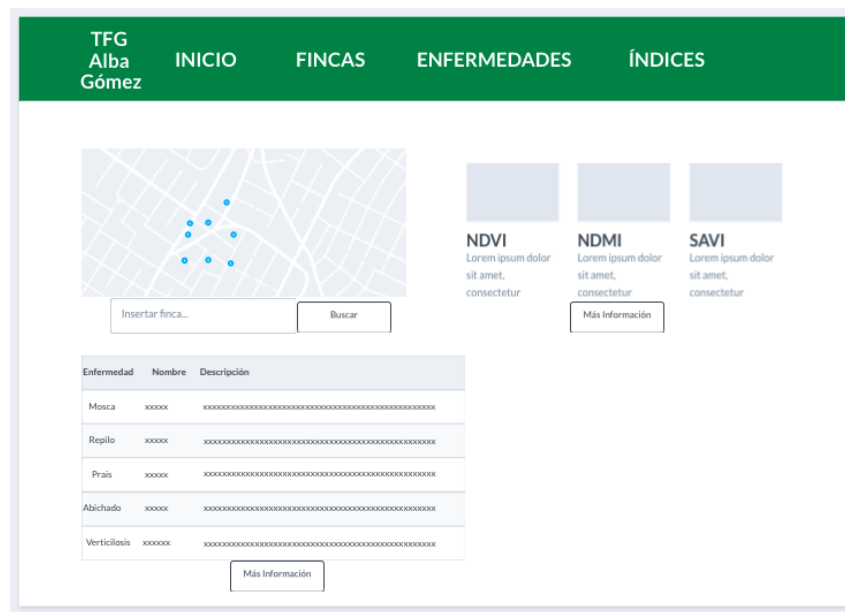


Figura 4.41: Mock-Up Página Principal - Elaboración Propia

- **Página Fincas (Fig 4.42):** Mock-up de la página donde se aparecerá el buscador de fincas y al final, aparecerán las más completas para mayor rapidez de búsqueda.



Figura 4.42: Mock-Up Página Fincas - Elaboración Propia

- **Página Finca (Fig 4.43):** Mock-up de la página específica de una finca, donde aparecerá toda la información de ella a la izquierda, a la derecha aparecerá un mapa con el contorno de la finca y si hay olivos disponibles y debajo todos los gráficos sobre ella.



Figura 4.43: Mock-Up Página Finca Específica - Elaboración Propia

- **Página Olivo (Fig 4.44):** Mock-up de la página específica de un olivo, aparecerá toda su información y las gráficas específicas de sus índices de vegetación.



Figura 4.44: Mock-Up Página Olivo Específico - Elaboración Propia

- **Página Índices de Vegetación (Fig 4.45):** Mock-up de la página donde se explicarán los diferentes índices de vegetación, por cada panel se explicará uno.



- **Página Tabla Enfermedades (Fig 4.46):** Mock-up de la página donde se explicarán las diferentes enfermedades más comunes del olivar en una tabla indicando el grado de incidencia de los posibles factores.



4.6.3. Implementación de la Interfaz de Usuario

En este apartado se va a explicar la implementación completa de la interfaz de usuario, para ello, se va a explicar de forma independiente cada una de las vistas que se han generado en el Mock-Up.

También me gustaría indicar que para la implementación de la página web se ha usado la siguiente Paleta Cromática de la Figura 4.47 para que la aplicación fuese lo más visual y atractiva posible.



Figura 4.47: Paleta de Colores Interfaz de Usuario - Elaboración Propia

Configuración Vue JS

Para la configuración de Vue JS en el proyecto junto con el Backend, en primer lugar, se ha establecido el puerto a 3000, mientras que el puerto de Spring es el 4000, mediante */TFG* se pondría *'http://localhost:4000'* como podemos ver en el siguiente Código 4.25.

```
1 import { fileURLToPath, URL } from 'node:url'
2 import { defineConfig } from 'vite'
3 import vue from '@vitejs/plugin-vue'
4
5 export default defineConfig({
6   plugins: [
7     vue(),
8   ],
9   resolve: {
10    alias: {
11      '@': fileURLToPath(new URL('./src', import.meta.url))
12    }
13  },
14  server: {
```



```
15   port: 3000,  
16   proxy: {  
17     '/TFG': {  
18       target: 'http://localhost:4000',  
19       ws: true,  
20       changeOrigin: true  
21     }  
22   }  
23 }  
24 })
```

Listado 4.25: Archivo vite.config.js

En el archivo *main.js* indicamos dónde se encuentra el router, bootstrap y la App en App.vue como podemos ver en el Código 4.26.

```
1  import { fileURLToPath, URL } from 'node:url'  
2  import { defineConfig } from 'vite'  
3  import vue from '@vitejs/plugin-vue'  
4  
5  export default defineConfig({  
6    plugins: [  
7      vue(),  
8    ],  
9    resolve: {  
10     alias: {  
11       '@': fileURLToPath(new URL('./src', import.meta.url))  
12     }  
13   },  
14   server: {  
15     port: 3000,  
16     proxy: {  
17       '/TFG': {  
18         target: 'http://localhost:4000',  
19         ws: true,  
20         changeOrigin: true  
21       }  
22     }  
23   }  
24 })
```

Listado 4.26: Archivo main.js

Por otro lado, en el archivo *router/index.js* en el Código 4.27, se configuran las rutas para nuestra aplicación usando Vue Router. Se importan las funciones *createRouter* y *createWebHistory*, así como varios componentes de vista. Luego, se crea un objeto de enrutador (router) utilizando *createRouter* con un historial de navegación basado en la API de historial HTML5 (*createWebHistory*). Cada ruta se asocia con un componente específico, como *HomeView* para la ruta principal (/) y *OlivoView* para la ruta */Olivo-View/:idObjeto* que acepta un parámetro de ruta (*:idObjeto*) y lo pasa al componente como propiedades (*props: true*).

```
1 import { createRouter, createWebHistory } from 'vue-router'
2 import HomeView from '../views/HomeView.vue'
3 import OlivoView from "../views/OlivoView.vue";
4
5 const router = createRouter({
6   history: createWebHistory(import.meta.env.BASE_URL),
7   routes: [
8     {
9       path: '/',
10      name: 'home',
11      component: HomeView
12    },
13    {
14      path: '/enfermedad',
15      name: 'enfermedad',
16      component: () => import('../views/EnfermedadView.vue')
17    },
18    {
19      path: '/finca',
20      name: 'finca',
21      component: () => import('../views/FincaView.vue')
22    },
23    {
24      path: '/indices',
25      name: 'indice',
26      component: () => import('../views/IndicesVegetacionView.vue')
27    },
28    {
29      path: '/fincaEsp',
30      name: 'fincaEsp',
31      component: () => import('../views/FincaEspecifica.vue')
32    },
33    {
34      path: '/OlivoView/:idObjeto',
35      name: 'OlivoView',
36      component: OlivoView,
37      props: true
38    }
39  ]
40 })
41
42 export default router
```

Listado 4.27: Archivo router/index.js

Implementación de la Página de Inicio

Como Vue.js está basado en componentes el primer archivo del que debemos hablar es *App.vue*, en ella, se configura la estructura de una página web utilizando *Vue Router*. En la sección *<script setup>*, se importan *RouterLink* y *RouterView* de *vue-router* para manejar la navegación y visualización de componentes basados en las rutas, junto con un componente personalizado *Footer*, que es el pie de nuestra

página web.

En la sección *template* se crea una barra de navegación personalizada con links a las vistas de Fincas, Enfermedades e Índices de Vegetación. Este componente principal nos sirve como base para la aparición de los siguientes. El código se encuentra en 4.28. Indicar que la parte de estilo se va a obviar pues es CSS básico.

```

1 <script setup>
2 import { RouterLink, RouterView } from 'vue-router'
3 import Footer from '@/components/Footer.vue'
4 </script>
5
6 <template>
7   <header>
8     <nav class="navbar navbar-expand-lg navbar-dark">
9       <div class="container-fluid">
10         <router-link class="navbar-brand" to="/"></router-link>
11         <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-
            target="#navbarNavDropdown" aria-controls="navbarNavDropdown" aria-expanded="
            false" aria-label="Toggle navigation">
12           <span class="navbar-toggler-icon"></span>
13         </button>
14         <div class="collapse navbar-collapse" id="navbarNavDropdown">
15           <ul class="navbar-nav">
16             <li class="nav-item">
17               <router-link class="nav-link active" aria-current="page" to="/">Inicio</
                router-link>
18             </li>
19             <li class="nav-item">
20               <router-link class="nav-link" to="/finca">Fincas</router-link>
21             </li>
22             <li class="nav-item">
23               <router-link class="nav-link" to="/enfermedad">Enfermedades</router-link>
24             </li>
25             <li class="nav-item">
26               <router-link class="nav-link" to="/indices">Índices de vegetación</router-
                link>
27             </li>
28           </ul>
29         </div>
30       </div>
31     </nav>
32   </header>
33
34   <main>
35     <RouterView />
36   </main>
37   <Footer />
38 </template>

```

Listado 4.28: Archivo App.vue

El siguiente paso es definir el componente HomeView.js, este se corresponde con la vista de inicio de la aplicación como podemos ver en la Figura 4.48 . En esta vista,

definimos un mapa donde se mostrarán las distintas fincas almacenadas en la base de datos, para eso usamos la etiqueta personalizada `<LeafletMap />`. Tenemos también un buscador donde podemos introducir el código SigPac y nos redirigirá a la página de la finca si existe, si no, muestra errores personalizados si no está en la BD o si el formato es incorrecto. Para ello, hacemos un *fetch* a la ruta que aparece en el método *buscarFinca()*.

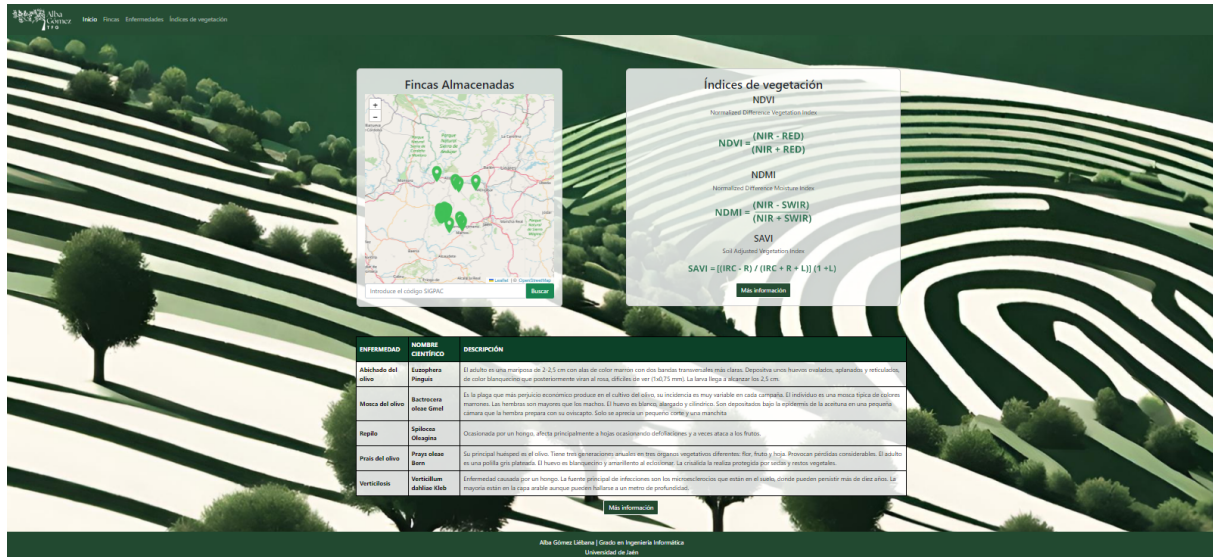


Figura 4.48: Página Home de la Aplicación Web - Elaboración Propia

A modo de introducción, mediante `<IndicesPeq />` y `<EnfermedadesPeq />` se muestra un resumen de lo que se puede ver en esas vistas, que se explicarán en apartados posteriores, ambas tendrán un botón de más información que redirige a esas vistas.

```

1 <template>
2   <div class="container">
3     <div class="content">
4       <div class="search-form-container">
5         <form @submit.prevent="buscarFinca" class="search-form">
6           <LeafletMap />
7           <div class="input-group">
8             <input v-model="codigoSIGPAC" type="text" class="form-control" placeholder="
9               Introduce el código SIGPAC">
10             <button type="submit" class="btn btn-success"><i class="bi bi-search"></i> Buscar
11             </button>
12           </div>
13           <div v-if="mostrarErrorFormato" class="error-message">
14             Formato SIGPAC no válido (provincia:municipio:agregado:zona:polígono:parcela:
15               recinto)
16           </div>
17           <div v-if="mostrarErrorEncontrado" class="error-message">
18             Código SIGPAC no encontrado.
19           </div>
20         </form>
21       </div>
22       <div class="info-ind">
23         <IndicesPeq />
24       </div>
25     </div>
26   </div>

```

```

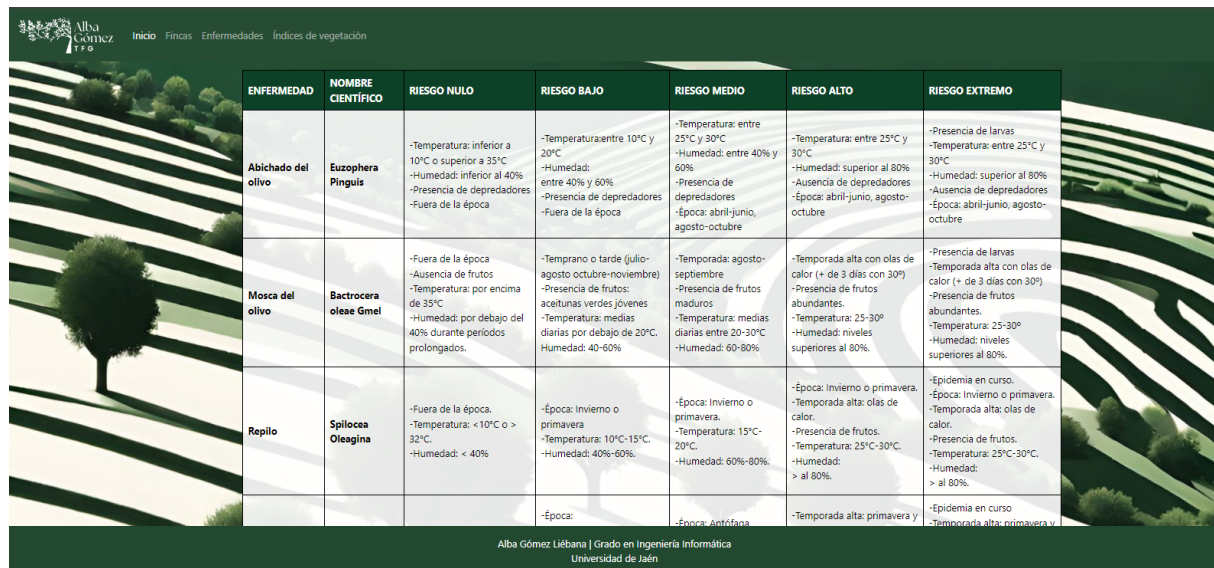
21     </div>
22     </div>
23     <div class="info-enf">
24         <EnfermedadesPeq />
25     </div>
26     <main>
27     </main>
28 </div>
29 </template>
30
31 <script>
32 ...
33     methods:
34     {
35         buscarFinca()
36         {
37             const formatoValido = /^[0-9]+:[0-9]+:[0-9]+:[0-9]+:[0-9]+:[0-9]+:[0-9]+$/ .test(
38                 this.codigoSIGPAC);
39             const [provinciaCodigo, municipioCodigo, , , poligono, parcela, recinto] = this.
40                 codigoSIGPAC.split(':');
41             fetch(`/TFG/provincia/${provinciaCodigo}/municipio/${municipioCodigo}/finca/${
42                 poligono}/${parcela}/${recinto}`)
43                 .then(response =>
44                 {
45                     if (!response.ok) {
46                         throw new Error('HTTP error! status: ${response.status}');
47                     }
48                     return response.json();})
49                 .then(data => {
50                     this.$store.commit('setFinca', data);
51                     this.$router.push({name: 'fincaEsp'});
52                 })
53                 .catch(error => {
54                     console.error('Error en la solicitud:', error);
55                     if (!formatoValido)
56                     {
57                         this.mostrarErrorFormato = true;
58                         this.mostrarErrorEncontrado = false;
59                     }
60                     else
61                     {
62                         this.mostrarErrorFormato = false;
63                         this.mostrarErrorEncontrado = true;
64                     }
65                 });
66         }
67     }
68 };
69 </script>
70 ...

```

Listado 4.29: Archivo HomeView.vue

Implementación de la Página de Enfermedades

La página donde se muestran las diferentes enfermedades la podemos ver en la Figura 4.49, en ella se han definido dos vectores de datos, *headers* para los encabezados de la tabla y *data* para toda la información sobre las enfermedades. En el código simplemente se muestra la tabla como podemos ver en el Código 4.30.



ENFERMEDAD	NOMBRE CIENTÍFICO	RIESGO NULO	RIESGO BAJO	RIESGO MEDIO	RIESGO ALTO	RIESGO EXTREMO
Abichado del olivo	Euzophera Pinguis	-Temperatura: inferior a 10°C o superior a 35°C -Humedad: inferior al 40% -Presencia de depredadores -Fuera de la época	-Temperatura: entre 10°C y 20°C -Humedad: entre 40% y 60% -Presencia de depredadores -Fuera de la época	-Temperatura: entre 25°C y 30°C -Humedad: entre 40% y 60% -Presencia de depredadores -Época: abril-junio, agosto-octubre	-Temperatura: entre 25°C y 30°C -Humedad: superior al 80% -Ausencia de depredadores -Época: abril-junio, agosto-octubre	-Presencia de larvas -Temperatura: entre 25°C y 30°C -Humedad: superior al 80% -Ausencia de depredadores -Época: abril-junio, agosto-octubre
Mosca del olivo	Bactrocera oleae Gmel	-Fuera de la época -Ausencia de frutos -Temperatura: por encima de 35°C -Humedad: por debajo del 40% durante periodos prolongados.	-Temprano o tarde (julio-agosto octubre-noviembre) -Presencia de frutos: aceitunas verdes jóvenes -Temperatura: medias diarias por debajo de 20°C. Humedad: 40-60%	-Temporada: agosto-septiembre -Presencia de frutos maduros -Temperatura: medias diarias entre 20-30°C -Humedad: 60-80%	-Temporada alta con olas de calor (+ de 3 días con 30°) -Presencia de frutos abundantes. -Temperatura: 25-30° -Humedad: niveles superiores al 80%.	-Presencia de larvas -Temporada alta con olas de calor (+ de 3 días con 30°) -Presencia de frutos abundantes. -Temperatura: 25-30° -Humedad: niveles superiores al 80%.
Repilo	Spilotea Oleagina	-Fuera de la época. -Temperatura: <10°C o >32°C. -Humedad: < 40%	-Época: Invierno o primavera -Temperatura: 10°C-15°C. -Humedad: 40%-60%.	-Época: Invierno o primavera. -Temperatura: 15°C-20°C. -Humedad: 60%-80%.	-Época: Invierno o primavera. -Temporada alta: olas de calor. -Presencia de frutos. -Temperatura: 25°C-30°C. -Humedad: > al 80%.	-Epidemia en curso. -Época: Invierno o primavera. -Temporada alta: olas de calor. -Presencia de frutos. -Temperatura: 25°C-30°C. -Humedad: > al 80%.
			-Época:	-Época: Antífona	-Temporada alta: primavera y	-Epidemia en curso -Temporada alta: primavera y

Figura 4.49: Página Enfermedades de la Aplicación Web - Elaboración Propia

```

1 <template>
2   <div class="table-container">
3     <table class="custom-table">
4       <thead>
5         <tr>
6           <th v-for="header in headers" :key="header" style="font-weight: bold; background-
7             color: #0c4128; color: white;">{{ header }}</th>
8         </tr>
9       </thead>
10      <tbody>
11        <tr v-for="(row, index) in data" :key="index">
12          <td v-for="(cell, cellIndex) in row" :key="cellIndex" style="color: black;">
13            <strong v-if="cellIndex < 2">{{ cell }}</strong>
14            <span v-else v-html="cell.replace(/\n/g, '<br>')"></span>
15          </td>
16        </tr>
17      </tbody>
18    </table>
19  </div>
20 </template>

```

Listado 4.30: Archivo EnfermedadView.vue

Implementación de la Página de los Índices de Vegetación

Esta página, al igual que la anterior, es también bastante sencilla, declaramos un vector con los diferentes campos de los índices (nombre, nombre completo, descripción y rango) e importamos el componente *IndiceVegetacion.vue*. En el apartado *template*, se llamará a este componente con cada uno de los índices para que los muestre en pantalla formateados. El resultado se puede ver en la Figura 4.50.

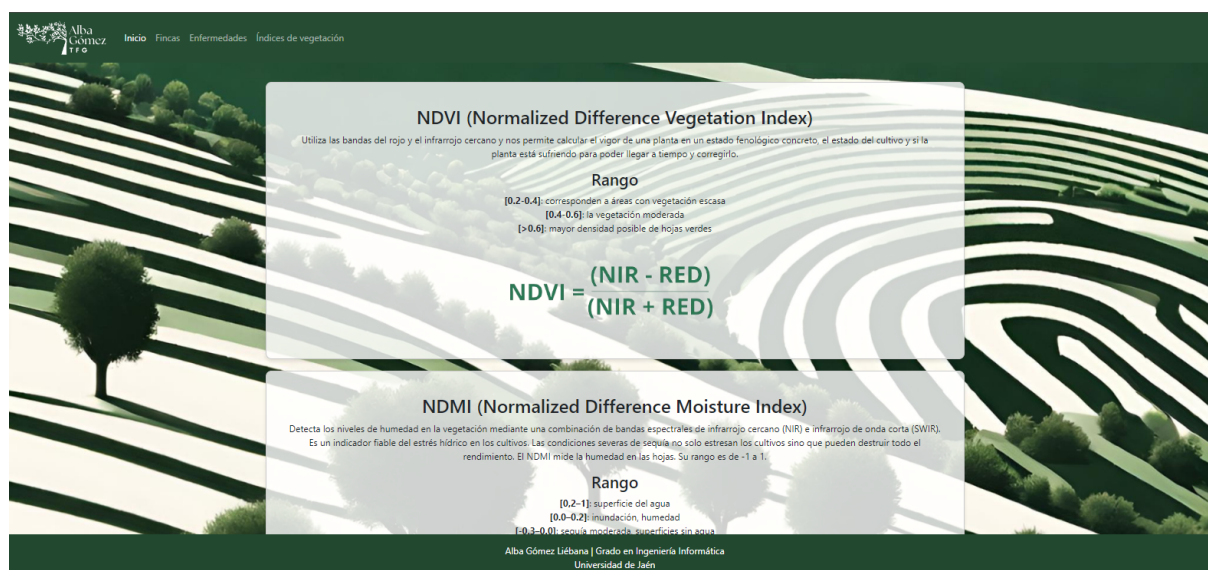


Figura 4.50: Página Índices de Vegetación de la Aplicación Web - Elaboración Propia

Implementación de la Página de las Fincas

En esta vista, representada en la Figura 4.51, ya se empieza a hacer evidente la complejidad de la página web, podemos ver el código más representativo en 4.31. En primer lugar, se muestra un formulario similar al que aparece en la página de inicio 4.6.3. Después, aparecería un listado de todas las fincas que tienen registrado histórico de datos, de esta forma, agilizamos el proceso de búsqueda. Para ello, debemos acceder a la ruta */TFG/FincasConHistorico*, las ordenamos alfabéticamente y se muestran en pantalla a través del componente *Finca*, al que le mandamos cada una de las fincas y el archivo geojson para mostrar su contorno.

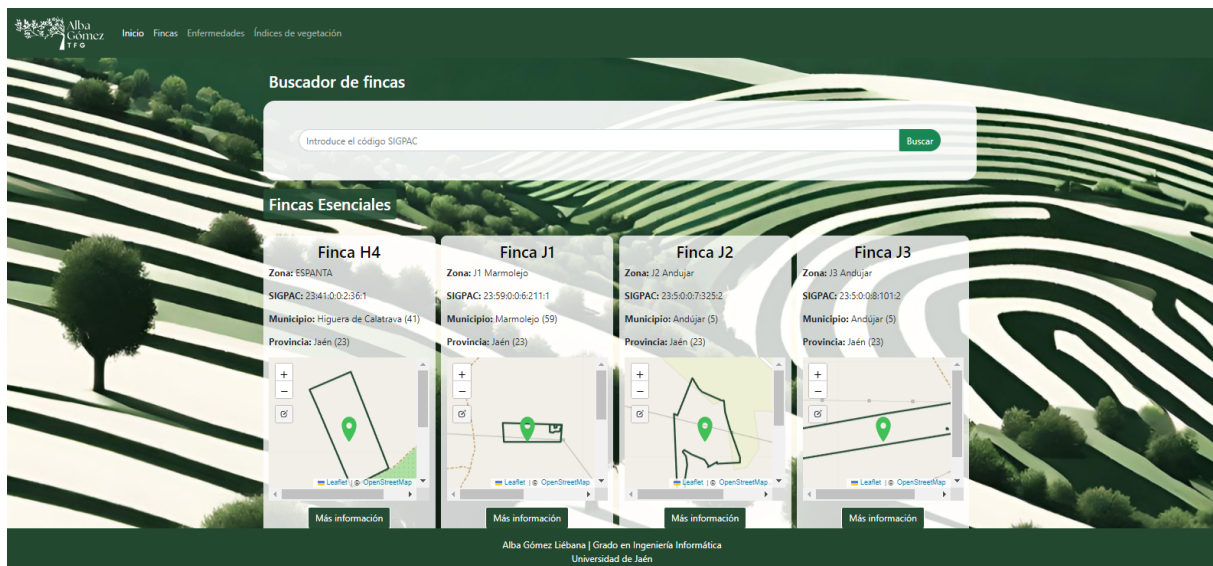


Figura 4.51: Página Buscador Fincas de la Aplicación Web - Elaboración Propia

```

1 <template>
2   ... //El código del buscador es igual que el de la página de inicio
3   <h3 class="subtitulo">Fincas Esenciales</h3>
4   <div class="grid-container">
5     <div v-for="finca in fincas" :key="finca.id" class="grid">
6       <Finca :finca="finca" :zonaUrl="getZonaUrl(finca.zonaUbicacion)" />
7     </div>
8   </div>
9 </template>
10
11 <script>
12 import Finca from '../components/FincaSIGPAC.vue';
13
14 ...
15 mounted() {
16   fetch('/TFG/FincasConHistorico')
17     .then(response => response.json())
18     .then(data => {
19       this.fincas = data.sort((a, b) =>
20         {
21           if (a.zonaUbicacion < b.zonaUbicacion)
22             {
23               return -1;
24             }
25           if (a.zonaUbicacion > b.zonaUbicacion)
26             {
27               return 1;
28             }
29           return 0;
30         });
31     });
32 },
33 methods: {
34   getZonaUrl(zonaUbicacion)
35   {
36     return '../assets/shp/${zonaUbicacion}.geojson';
37   },

```



```

38 |     ...
39 | };
40 | </script>

```

Listado 4.31: Archivo FincaView.vue

Para cada una de las fincas a mostrar, el componente *FincaSIGPAC.vue* muestra su zona, su código SIGPAC, municipio, provincia y un mapa donde aparecería el contorno de la misma. Además, tenemos un botón de más información que nos redirige a la misma vista que si hubiéramos buscado su código en el buscador. Como es más intuitivo mostrar el nombre del municipio y la provincia, además del código, se acceden a las distintas rutas implementadas en el servicio REST que nos devuelven los datos solicitados, podemos verlo en el Código 4.32.

```

1 | <template>
2 |   <div class="finca-details">
3 |     <div v-if="finca_{{Object.keys(zona).length}}>
4 |       <h3>Finca {{finca.zonaUbicacion}}</h3>
5 |       <p><strong>Zona:</strong> {{zona.descripcion}}</p>
6 |       <p><strong>SIGPAC:</strong> {{ finca.provinciaCodigo }}:{{ finca.municipioCodigo
7 |         }}:0:0:{{ finca.poligono }}:{{ finca.parcela }}:{{ finca.recinto }} </p>
8 |       <p><strong>Municipio:</strong> {{municipio.nombre}} ({{ finca.municipioCodigo }})
9 |         </p>
10 |      <p><strong>Provincia:</strong> {{provincia.nombreProvincia}} ({{ finca.
11 |        provinciaCodigo }})</p>
12 |      <FincaMapa :latitud="zona.latitud" :longitud="zona.longitud" :zona="finca.
13 |        zonaUbicacion"/><div class="button-container">
14 |       <button @click="verMasInformacion" class="btn btn-primary button-custom">Más
15 |         información</button>
16 |     </div>
17 |   </div>
18 |   <div v-else>
19 |     <p>Cargando datos de la finca...</p>
20 |   </div>
21 | </template>
22 |
23 | <script>
24 | ...
25 |   watch: {
26 |     finca: {
27 |       immediate: true,
28 |       handler(newFinca)
29 |       {
30 |         if (newFinca)
31 |         {
32 |           fetch('/TFG/provincias/${newFinca.provinciaCodigo}/municipios/${newFinca.
33 |             municipioCodigo}/zonas/${newFinca.zonaUbicacion}')
34 |             .then((response) => response.json())
35 |             .then((data) => {

```

```

36         this.provincia=data;
37     });
38     fetch(`/TFG/provincias/${newFinca.provinciaCodigo}/municipios/${newFinca.
        municipioCodigo}`)
39         .then((response) => response.json())
40         .then((data) => {
41             this.municipio=data;
42         });
43     ...

```

Listado 4.32: Archivo FincaSIGPAC.vue

Por último, se va a explicar el componente *FincaMapa.vue* en el Código 4.33, este utiliza la biblioteca *Leaflet* para mostrar un mapa interactivo. Primero, se importan varias funciones de Vue y Leaflet junto con sus hojas de estilo. Se definen las propiedades latitud, longitud y zona que se esperan recibir como *props*. Se crea una referencia *map* para almacenar el mapa y se inicializa un marcador (*marker*) en *null*. Cuando el componente se monta (*onMounted*), se configura el mapa centrado en las coordenadas proporcionadas, se agrega una capa de *OpenStreetMap*, se inicializa el *plugin Leaflet Draw* para dibujar en el mapa, y se carga un archivo *GeoJSON* específico para delinear una zona en el mapa.

Además, se crea un icono personalizado para el marcador, se coloca en el mapa en las coordenadas especificadas y se vincula con un *popup*. La función *watch* se utiliza para observar los cambios en las propiedades latitud y longitud. Si cambian, el marcador se actualiza a las nuevas coordenadas y el mapa se centra en esta nueva ubicación.

```

1 <script setup>
2 ...
3 onMounted(async () => {
4     map.value = L.map(map.value).setView([props.latitud, props.longitud], 16);
5
6     // Agregar capa de OpenStreetMap
7     L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
8         maxZoom: 19,
9         attribution: '&copy; <a href="http://www.openstreetmap.org/copyright">OpenStreetMap </a>',
10    }).addTo(map.value);
11
12    // Inicializar el plugin Leaflet Draw
13    const drawnItems = new L.FeatureGroup();
14    map.value.addLayer(drawnItems);
15    const drawControl = new L.Control.Draw({
16        draw: {
17            polyline: false,
18            polygon: false,
19            circle: false,
20            rectangle: false,
21            marker: false,
22            circlemarker: false,

```

```

23   },
24   edit: {
25     featureGroup: drawnItems,
26     remove: false,
27   }
28 });
29 map.value.addControl(drawControl);
30
31 //PARA CARGAR EL CONTORNO DEL MAPA
32 let geojson;
33 try {
34   const module = await import('@assets/geojson/${props.zona}.json');
35   geojson = module.default;
36 } catch (error) {
37   console.error('Error al cargar el archivo GeoJSON:', error);
38   return;
39 }
40 L.geoJson(geojson, {
41   style: { "color": "#224930", "fill": false }
42 }).addTo(map.value);
43
44 const greenIcon = L.icon({
45   iconUrl: iconoUbicacion,
46   iconSize: [40, 45],
47 });
48
49 marker = L.marker([props.latitud, props.longitud], { icon: greenIcon }).addTo(map.value
50   ).bindPopup('Finca');
51 });
52 watch(() => [props.latitud, props.longitud], ([newLat, newLon]) => {
53   if (marker) {
54     marker.setLatLng([newLat, newLon]);
55     map.value.setView([newLat, newLon], 14);
56   }
57 });
58 </script>
59 ...

```

Listado 4.33: Archivo FincaMapa.vue

Implementación de la Página Finca Específica

Una vez se haya introducido un código SigPac válido en los buscadores de la pantalla de inicio o de las fincas o se haya pulsado a "Más Información" de alguna de las fincas que aparecen con histórico, se nos redirigirá a esta Página 4.52, es una de las más importantes de la aplicación junto con la siguiente Página 4.6.3.

Empecemos analizando el apartado template del componente 4.34. En primer lugar, se muestran los datos de la finca, incluyendo el nombre del municipio, provincia (que se ha hecho una petición REST, como se ha explicado antes), polígono, parce-

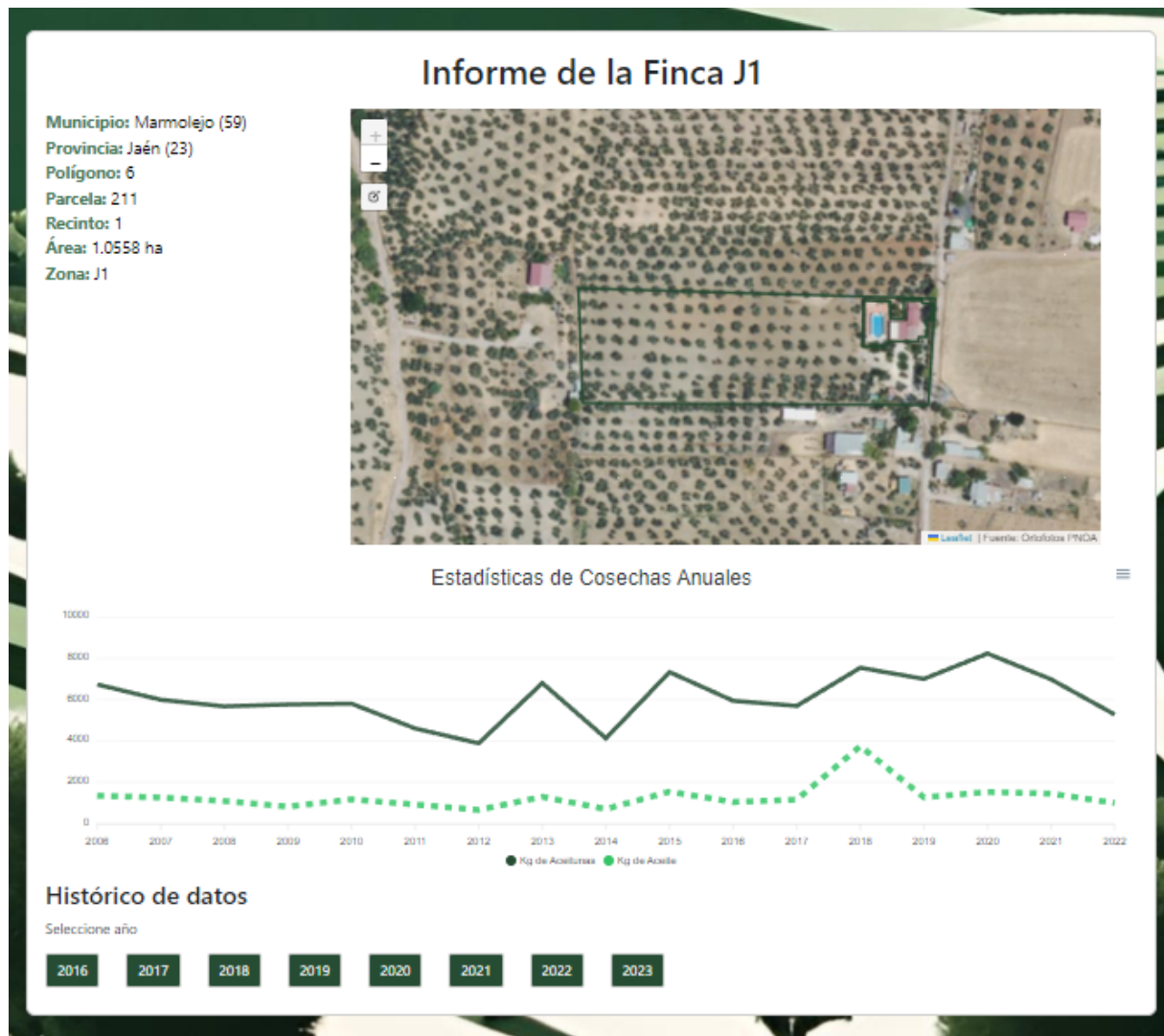


Figura 4.52: Página Finca Específica de la Aplicación Web - Elaboración Propia

la, recinto, área y la zona donde se ubica. A la derecha, aparecería un mapa de la parcela, que se comentará más adelante.

Tras lo mencionado anteriormente, aparecía una gráfica interactiva mostrando los kilogramos de aceituna recogidos desde que se tiene información de la finca, al igual que los kilogramos de aceite que se produjeron mediante una etiqueta llamada MapaPNOA.

Por último, aparecerán una serie de botones, desde el año 2016 hasta 2023 para mostrar el histórico de la finca si está disponible, al pulsar sobre alguno de ellos, se cargarán las gráficas de ese año.

```

1 <template>
2   <div class="container">
3     <div class="finca-details">
4       <h2>Informe de la Finca {{finca.zonaUbicacion }}</h2>

```

```

5 | <div v-if="finca">
6 |   <div class="containerDatos">
7 |     <p class="data">
8 |       <strong>Municipio:</strong> {{ municipio.nombre }} ({{ finca.municipioCodigo
9 |         }}) <br>
10 |       <strong>Provincia:</strong> {{ provincia.nombreProvincia }} ({{ finca.
11 |         codigoProvincia }}) <br>
12 |       <strong>Polígono:</strong> {{ finca.poligono }} <br>
13 |       <strong>Parcela:</strong> {{ finca.parcela }} <br>
14 |       <strong>Recinto:</strong> {{ finca.recinto }} <br>
15 |       <strong> rea :</strong> {{ finca.area }} ha <br>
16 |       <strong>Zona:</strong> {{ finca.zonaUbicacion }}
17 |     </p>
18 |     <MapaPNOA class="finca-mapa" :latitud="zona.latitud" :longitud="zona.longitud"
19 |       :zona="finca.zonaUbicacion" :olivos="olivos"/>
20 |   </div>
21 |   <div v-else>
22 |     <p>Cargando datos de la finca...</p>
23 |   </div>
24 |   <div>
25 |     <h3>Histórico de datos </h3>
26 |     <p>Seleccione año</p>
27 |     <button v-for="year in years" :key="year" @click="selectYear(year)" class="btn
28 |       btn-success btn-year
29 |       :class="{ 'selected': selectedYear === year }" >{{ year }}</button>
30 |   </div>
31 |   <IndiceGrafica v-if="selectedYear" :year="selectedYear" :finca="finca" />
32 |   ...
33 | </template>

```

Listado 4.34: Archivo FincaEspecific.vue

A continuación, se va a detallar cada uno de los componentes.

- **MapaPNOA:** este mapa, como se puede ver en la Figura 4.53, muestra una imagen de la finca que se está estudiando junto con su límite. Además, aunque no aparezca en pantalla, si se pulsa sobre alguno de sus olivos, cambiará a la página 4.6.3 con su correspondiente información.

En cuanto a su *script*, en primer lugar, estarían los *import*, necesitamos la librería *Leaflet* para crear el mapa, *proj4* para convertir coordenadas y *turf* para la función *puntoEnPolígono*, *punto y polígono*. Definimos la función de conversión de coordenadas:

```

1 | const utmProjection = '+proj=utm +zone=30 +datum=WGS84 +units=m +no_defs';
2 | const convertUTMToLatLng = (x, y) =>
3 | {
4 |   const [lon, lat] = proj4(utmProjection, 'EPSG:4326', [x, y]);
5 |   return [lat, lon];
6 | };

```

Listado 4.35: Función conversión de coordenadas



Figura 4.53: Ejemplo Mapa Finca - Elaboración Propia

En cuanto se termina de pintar la vista en la página web, se ejecutaría *onMounted()*, este fragmento de Código 4.36 inicializa un mapa de Leaflet centrado en las coordenadas especificadas con un nivel de zoom de 18. Añade una capa **WMS del Instituto Geográfico Nacional de España** que muestra ortoimágenes. Luego, intenta cargar un archivo GeoJSON específico de la finca que muestra el contorno de ella.

Tras ello, se crea un grupo para elementos dibujados, que serán los olivos (*drawnItems*) y los añade al mapa de *Leaflet*. Luego, configura un control de dibujo (*drawControl*) que permite editar los elementos del grupo pero desactiva la creación de nuevas formas. Finalmente, añade este control al mapa.

```

1  map.value = L.map('map').setView([props.latitud, props.longitud], 18);
2  L.tileLayer.wms("http://www.ign.es/wms-inspire/pnoa-ma", {
3    layers: 'OI.OrthoimageCoverage',
4    attribution: "Fuente: Ortofotos PNOA"
5  }).addTo(map.value);
6
7  let geojson;
8  try {
9    const module = await import('@assets/geojson/${props.zona}.json');
10   geojson = module.default;
11 } catch (error) {
12   {
13     console.error('Error al cargar el archivo GeoJSON:', error);
14     return;
15   }
16   geojsonLayer = L.geoJson(geojson, {

```



```

17     style: { "color": "#224930", "fill": false }
18   }).addTo(map.value);
19   const drawnItems = new L.FeatureGroup();
20   map.value.addLayer(drawnItems);
21   const drawControl = new L.Control.Draw({
22     draw: {
23       polyline: false,
24       polygon: false,
25       circle: false,
26       rectangle: false,
27       marker: false,
28       circlemarker: false,
29     },
30     edit: {
31       featureGroup: drawnItems,
32       remove: false,
33     }
34   });
35   map.value.addControl(drawControl);
36 }
37 });

```

Listado 4.36: Definición capas en Leaflet, onMounted()

En el Código 4.37, se presenta una función *watch()*. Cuando *props.olivos* cambia, primero se verifica si el objeto *map* tiene un valor (*map.value*). Si es así, elimina todos los marcadores actuales del mapa y vacía el array *markers.value*. Luego, para cada olivo en *newOlivos*, convierte las coordenadas UTM a latitud y longitud usando la función *convertUTMToLatLng*, definida anteriormente y a continuación, verifica si el punto convertido está dentro de cualquier polígono en *geojsonLayer* utilizando *pointInPolygon*.

Si el punto está dentro de un polígono, el código crea un marcador circular en la posición latitud/longitud del olivo usando *L.circle*, lo agrega al mapa, y añade un evento de clic al círculo que redirige al usuario a una vista detallada del olivo (*OlivoView*), que se explicará en el apartado 4.6.3.

```

1 watch(() => props.olivos, (newOlivos) =>
2 {
3   if (map.value)
4   {
5     markers.value.forEach(marker => map.value.removeLayer(marker));
6     markers.value = [];
7
8     newOlivos.forEach(olivo =>
9     {
10      const [lat, lon] = convertUTMToLatLng(olivo.puntoMedio.x, olivo.puntoMedio.y)
11      ;
12      const olivoPoint = point([lon, lat]);
13      const isInside = geojsonLayer.getLayers().some(layer =>
14      {
15        const olivoPolygon = polygon(layer.toGeoJSON().geometry.coordinates);
16        return pointInPolygon(olivoPoint, olivoPolygon);
17      });
18    });
19  }
20 }

```

```

17
18     if (isInside)
19     {
20         const circle = L.circle([lat, lon], {
21             color: "rgba(34,73,48,0.39)",
22             radius: 3
23         }).addTo(map.value).bindPopup('Olivo: ${olivo.idObjeto}');
24
25         circle.on('click', () => { router.push({ name: 'OlivoView', params: {
26             idObjeto: olivo.idObjeto }});});
27
28         markers.value.push(circle);
29     }
30 }
31 }, { immediate: true });

```

Listado 4.37: Función watch() en Mapa PNOA

- **CosechaGráfica (Código 4.38):** en este componente se presenta la primera de las gráficas, se trata de un mostrar un gráfico de líneas con estadísticas de cosechas anuales, en esta iteración se va a hablar del código, su significado lo veremos en la siguiente iteración. Para ello, se usa *vue3-apexcharts*, el componente recibe una propiedad finca, y define dos series de datos para "Kg de Aceitunas" y "Kg de Aceite".

Cuando el componente se monta, se llama al método *fetchCosechasData* para obtener los datos de cosechas, después se actualizan las series del gráfico con los datos obtenidos, y se configuran las categorías del eje x con los años de las campañas.

```

1  ...
2  data() {
3      return {
4          series: [{
5              name: "Kg_de_Aceitunas",
6              data: []
7          },
8          {
9              name: "Kg_de_Aceite",
10             data: []
11          }
12      ],
13      chartOptions: {
14          chart: {
15              height: 350,
16              type: 'line',
17              zoom: {
18                  enabled: false
19              },
20          },
21          dataLabels: {
22              enabled: false
23          },

```



```

24    ...
25    mounted() {
26        this.fetchCosechasData();
27    },
28    methods: {
29        async fetchCosechasData() {
30            try {
31                const url = `TFG/provincia/${this.finca.codigoProvincia}/municipio/${this.
                    finca.municipioCodigo}/finca/${this.finca.poligono}/${this.finca.
                    parcela}/${this.finca.recinto}/cosechas`;
32                const response = await fetch(url);
33                let data = await response.json();
34                data = data.sort((a, b) => a.campania - b.campania);
35
36                const categories = [];
37                data.forEach(entry => {
38                    this.series[0].data.push(Math.round(entry.aceitunas));
39                    this.series[1].data.push(Math.round(entry.aceite));
40                    categories.push(entry.campania);
41                });
42
43                this.chartOptions = {
44                    ...this.chartOptions,
45                    xaxis: {
46                        ...this.chartOptions.xaxis,
47                        categories: categories
48                    }

```

Listado 4.38: Script CosechaGrafica

- **InsertarGráficosFinca:** este componente se encarga de presentar y gestionar datos históricos asociados a la finca seleccionada. En el apartado *template*, dispone de un botón que activa la función *fetchHistorico* al ser presionado, permitiendo cargar los datos del histórico del año.

Además, muestra un componente personalizado llamado *Gráficos* si existen datos disponibles en el histórico. En su lógica, el método *fetchHistorico* realiza solicitudes *fetch* a dos URLs diferentes para obtener datos históricos relacionados con la reflectancia, temperatura y lluvia. Luego, procesa los datos recibidos, los organiza y los asigna a vectores correspondientes para su posterior visualización. Cada conjunto de datos se convierte del formato codificado a JSON antes de ser almacenado. Además, se implementa un manejo de errores para situaciones donde la respuesta no es un arreglo o cuando los datos recibidos no siguen el formato esperado.

- **GráficosFinca:** este último componente, muestra los gráficos de los diferentes índices de vegetación, temperatura y lluvias 4.54. Todos los datos se reciben del anterior componente. Los gráficos se actualizan dinámicamente cuando cambian los datos de entrada. Cuando los datos aún no se han cargado, se muestra un mensaje de "Cargando datos..."

La función *prepareChartData* se encarga de preparar los datos necesarios para los gráficos, procesando los datos de entrada y actualizando las series y opciones de gráficos en consecuencia. Los datos se organizan en series y categorías, que se utilizan para mostrar los valores en los ejes x de los gráficos. Las categorías se obtienen de los datos de temperatura, mientras que las series se construyen a partir de los datos de reflectancia y otros datos meteorológicos.

Se muestran tanto datos a nivel satelital como de dron, si se tiene. Cada uno aparecería en una línea en los diferentes gráficos. Por ejemplo, para los datos del dron, se han creado dos vectores ya que sólo tenemos datos de NDVI y SAVI. Para cada uno de los datos, por tanto, se obtendrían los valores y se almacenarían en su array correspondiente y este luego se incluye a las series del gráfico 4.39.

Histórico de datos

Seleccione año

2016

2017

2018

2019

2020

2021

2022

2023

Cargar Datos

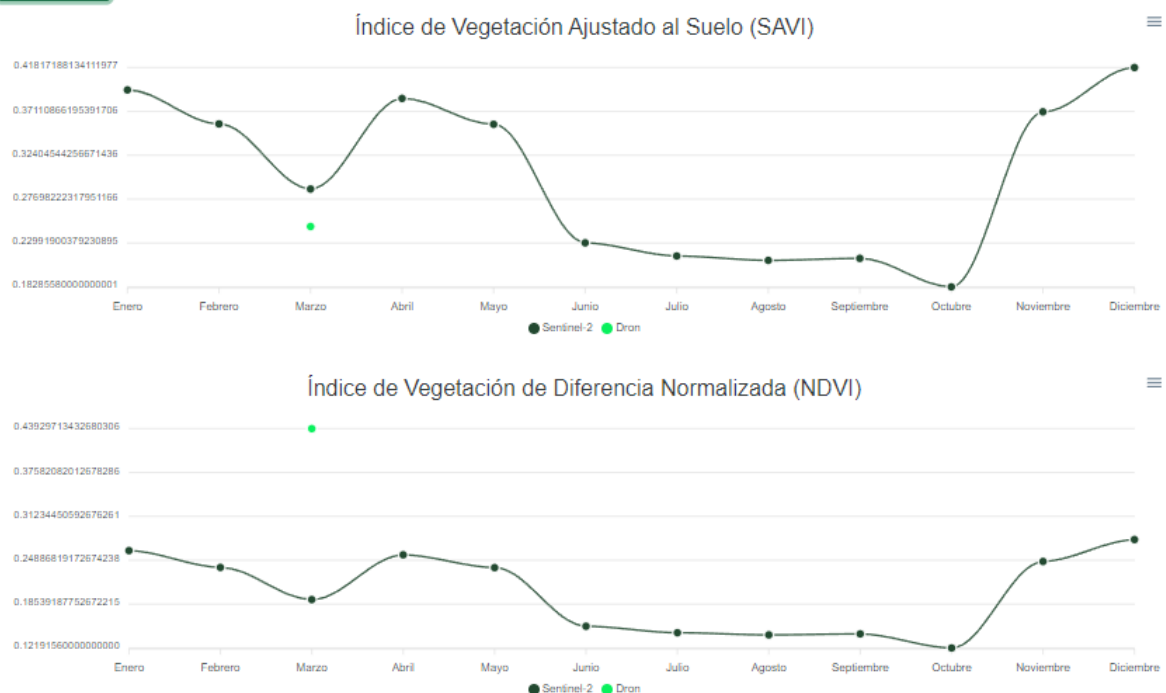


Figura 4.54: Ejemplo Gráficas Finca - Elaboración Propia

```

1  const allMonths = ['Enero', 'Febrero', 'Marzo', 'Abril', 'Mayo', 'Junio', '
2  Julio', 'Agosto', 'Septiembre', 'Octubre', 'Noviembre', 'Diciembre'];
3  const dataPerMonthDNDVI = new Array(12).fill(null);
4  const dataPerMonthDSAVI = new Array(12).fill(null);
5
6  this.reflectanciaDataD.forEach(item =>
7  {

```

```

8      if (item && item.data)
9      {
10         const ndviDron = item.data.find(subItem => Object.keys(subItem)[0] === '
            NDVI');
11         const saviDron = item.data.find(subItem => Object.keys(subItem)[0] === '
            SAVI');
12         if (ndviDron)
13         {
14             const value = typeof Object.values(ndviDron)[0] === 'string' ?
                parseFloat(Object.values(ndviDron)[0].replace(',', '.')) : Object.
                values(ndviDron)[0];
15             const monthIndex = allMonths.indexOf(item.mes);
16             if (monthIndex !== -1)
17             {
18                 dataPerMonthDNDVI[monthIndex] = value > 0 ? value : null;
19             }
20         }
21         if (saviDron)
22         {
23             const value = typeof Object.values(saviDron)[0] === 'string' ?
                parseFloat(Object.values(saviDron)[0].replace(',', '.')) : Object.
                values(saviDron)[0];
24             const monthIndex = allMonths.indexOf(item.mes);
25             if (monthIndex !== -1)
26             {
27                 dataPerMonthDSAVI[monthIndex] = value > 0 ? value : null;
28             }
29         }
30     }
31 });
32
33 NDVISeries.push({
34     name: "Dron",
35     data: dataPerMonthDNDVI,
36 });
37
38 SAVISeries.push({
39     name: "Dron",
40     data: dataPerMonthDSAVI,
41 });

```

Listado 4.39: Script Gráficos Finca

Implementación de la Página Olivo Específico

Cuando se pulsa sobre un olivo específico, la página a mostrar tendría el aspecto que podemos ver en la Figura 4.55. Arriba, aparecerán todos los datos del mismo, para ello se hace una petición REST y se obtienen todos sus datos y para el tema de los gráficos, se ha hecho igual que para la finca. Aparecerán los años a seleccionar y si existe histórico, aparece un botón de "Cargar Gráficos", al pulsar, aparecerán siguiendo la misma metodología que el apartado anterior, teniendo en cuenta que esta vez solo aparecerán los índices de vegetación.

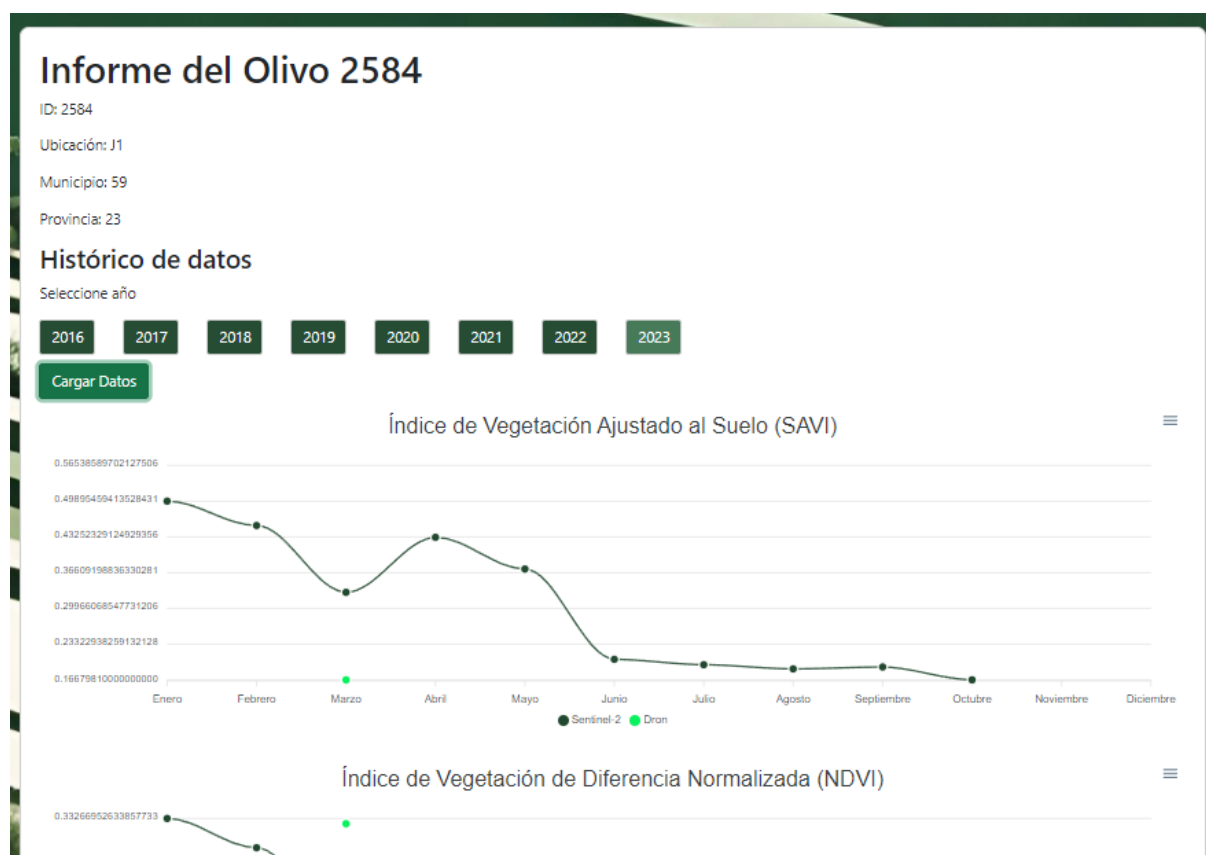


Figura 4.55: Ejemplo Vista Olivo - Elaboración Propia

4.7. Quinta iteración: Análisis de los Datos

En esta iteración, se va a explicar el análisis de los datos mostrados en las fincas que se añadieron en la aplicación en la iteración anterior [4.6.3](#).

4.7.1. Análisis Datos Finca

Se va a empezar con el análisis de los datos de las fincas, como ejemplo, vamos a tomar la Finca J1, situada en Marmolejo y con código SigPac 23:59:0:0:6:211:1. En la Figura [4.56](#) podemos ver su informe.

La primera Gráfica [4.57](#) que vemos se corresponde con las cosechas anuales, en este caso, tendríamos datos desde 2006 hasta 2022, se ha tomado como ejemplo 2021, en primer lugar podríamos ver los kilogramos de aceitunas seguidos de los de aceite.

Seguimos analizando la finca seleccionando un año para mostrar el histórico de datos, por ejemplo, se seleccionará 2023, que tiene datos tanto de dron como de

Informe de la Finca J1

Municipio: Marmolejo (59)
 Provincia: Jaén (23)
 Polígono: 6
 Parcela: 211
 Recinto: 1
 Área: 1.0558 ha
 Zona: J1



Figura 4.56: Informe Finca J1 - Elaboración Propia

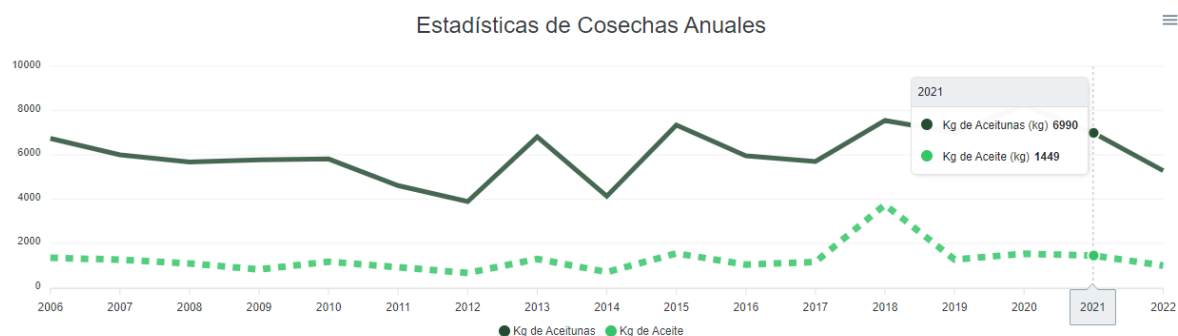


Figura 4.57: Gráfica Cosechas J1 - Elaboración Propia

satélite.

Los tres primeros gráficos 4.58 se corresponden con los Índices de Vegetación, tanto SAVI como NDVI tendrían disponibles datos de dron. Estos, aparecen en color verde clarito mientras que los satelitales de verde oscuro.

Podemos apreciar que hay más diferencia entre el dato de dron en NDVI que en SAVI, NDVI es un indicador que señala el verdor, por mencionarlo así, en cada píxel de una imagen, por tanto, el valor del satélite es la media de los píxeles de la imagen completa, para los de dron, sin embargo, se ha tomado la media de las imágenes de los olivos. Si analizamos el rango, el valor del dron (aproximadamente 0.43), correspondería a áreas de vegetación moderada y el de Sentinel-2 (0.21) a áreas con vegetación escasa. Vemos, por tanto, que el dato del dron es más acertado.

En cuanto al índice SAVI, no habría tanta diferencia, apenas 4 centésimas, en este índice, se incluye un factor de ajuste de suelo L para corregir los efectos del ruido del

suelo, por lo que tendría sentido.

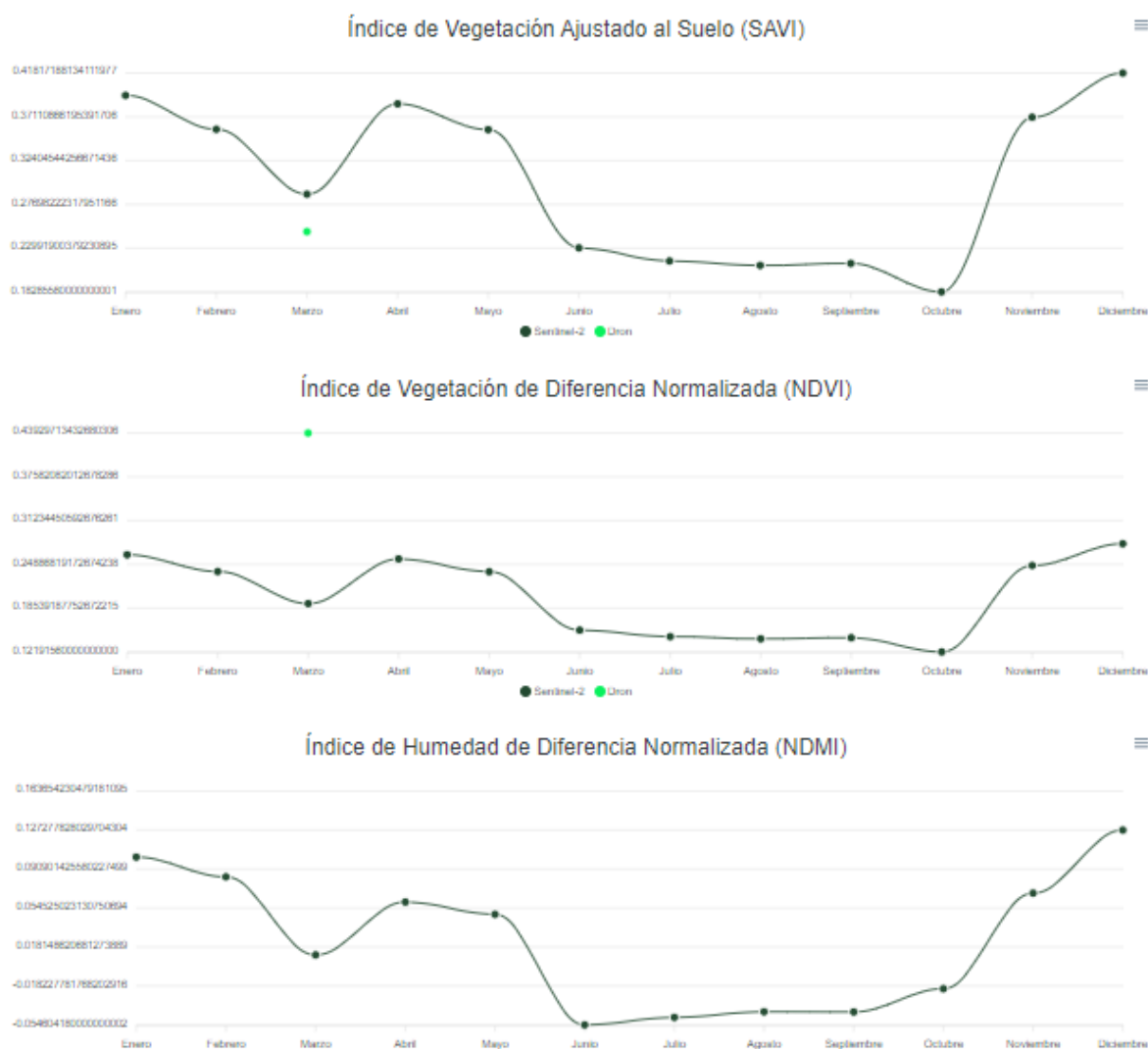


Figura 4.58: Gráficos Índices de Vegetación J1 - Elaboración Propia

Seguimos con el gráfico de precipitaciones registradas [4.59](#), vemos que las épocas donde más llovió ese año fue en mayo y en octubre, correspondiendo con primavera y otoño. Este gráfico puede ser útil para planificar el riego y otras actividades agrícolas, adaptándolas según las condiciones climáticas. Además, permite identificar períodos de sequía o exceso de lluvia, lo que influye en decisiones como la siembra y la cosecha.

Por último, tendríamos el gráfico de las temperaturas registradas, en ella, como se ve en la Figura [4.60](#), se observa las temperaturas mínima, máxima y media, tanto del día como la noche agrupadas por mes. Esto proporciona una visión detallada de las variaciones climáticas a lo largo del año. En cuanto a las utilidades, se puede ajustar el calendario de siembra y cosecha según las temperaturas esperadas. Además, este tipo de gráfico ayuda a identificar patrones climáticos, como períodos de heladas o de



Figura 4.59: Gráfico Precipitaciones J1 - Elaboración Propia

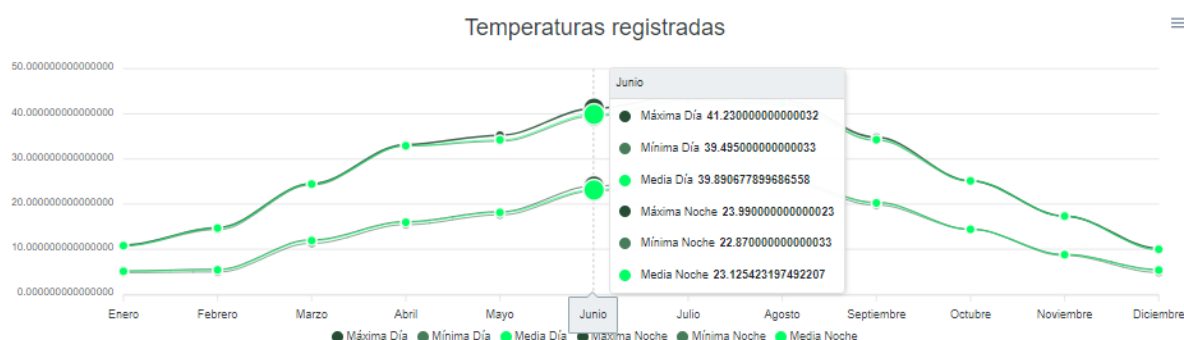


Figura 4.60: Gráfico Temperaturas J1 - Elaboración Propia

altas temperaturas, que pueden afectar el crecimiento y desarrollo de los cultivos.

4.7.2. Análisis Datos Olivo

Para este ejemplo se va a analizar el olivo 2492 correspondiente a la finca J1. Para el histórico de datos, se seleccionará el año 2023, dónde se podran observar todos los datos agrupados por meses.

Poder obtener gráficos específicos de cada olivo de la finca puede ser extremadamente útil. En primer lugar, estos gráficos proporcionan información detallada sobre la salud y el vigor de cada olivo, lo que permite a los agricultores identificar áreas problemáticas, como estrés hídrico o deficiencias nutricionales, y tomar medidas correctivas específicas, como ajustar el riego o aplicar fertilizantes.

Además, estos gráficos pueden ayudar a monitorear el crecimiento y desarrollo de los olivos a lo largo del tiempo, lo que facilita la evaluación del rendimiento y la planificación de la cosecha. Además, al comparar los índices de vegetación de diferentes olivos, los agricultores pueden identificar patrones y tendencias que pueden influir en

la gestión general del olivar.

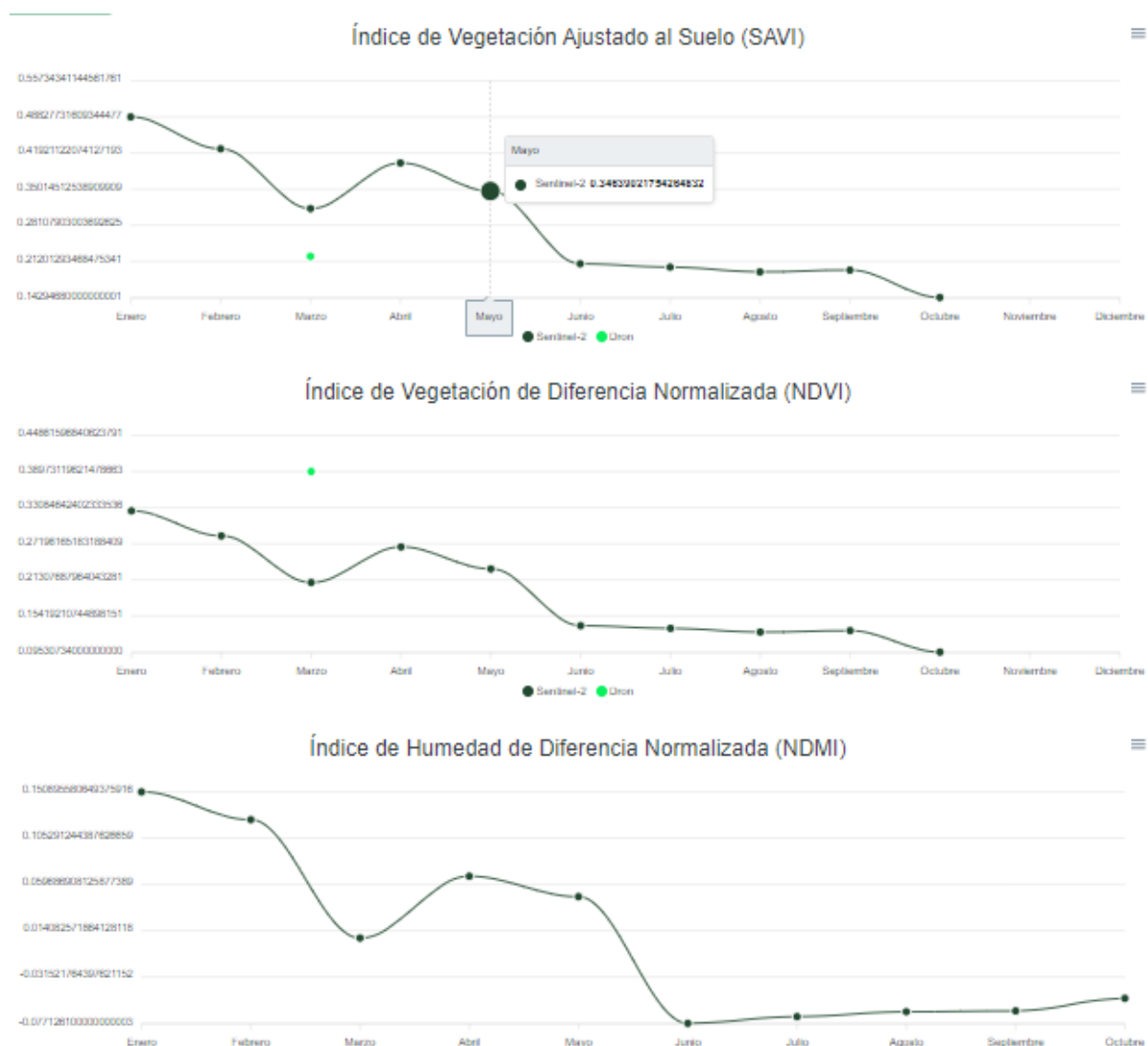


Figura 4.61: Ejemplo Gráficos Índices de Vegetación Olivo - Elaboración Propia

En la Figura 4.61 podemos ver los tres gráficos correspondientes a los índices de vegetación. De nuevo, se cumple el patrón explicado en el apartado anterior, la diferencia más clara se da en el índice NDVI, mientras que el de satélite muestra una zona con vegetación escasa, el de dron la identifica como moderada. El índice SAVI, por otro lado, engloba los dos valores en el mismo rango.

4.8. Pruebas finales con usuarios

En este apartado, se van a incluir las diferentes opiniones de distintos perfiles en cuanto a la aplicación web desarrollada. Se han elaborado dos cuestionarios diferen-

tes, uno para un perfil más informático y otro para un perfil de técnico agrícola, a continuación se muestran los resultados.

4.8.1. Resultados perfil informático

Se ha consultado a dos ingenieros informáticos, Álvaro de Castro Escribano y Luis René Domínguez Fuentes. Han expresado opiniones positivas sobre la página web, destacando su diseño bonito y fácil de entender. La consideran una herramienta con mucho potencial, especialmente relevante en el contexto local de Jaén y la industria del olivar. Resaltan el seguimiento de diferentes parámetros sobre varias fincas. Además, valoran los apartados que explican los tipos de índices de vegetación, el cálculo de estos índices, y las enfermedades y plagas comunes en estos cultivos, facilitando la comprensión y el uso de la aplicación para cualquier persona.

- **Opiniones sobre la interfaz de usuario:** ambos usuarios consideran la interfaz intuitiva y sencilla, lo que facilita su uso tanto para usuarios experimentados como para novatos. También mencionan que la interfaz es clásica y tiene una paleta de colores bien definida.
- **Respuesta de la página web a las interacciones del usuario:** ambos usuarios coinciden en que la página web responde de manera rápida y eficiente a las interacciones del usuario, aunque uno sugiere evaluar el rendimiento al manejar más datos y históricos. Ninguno de los usuarios ha notado errores o problemas técnicos mientras exploraban la página web.
- **Sugerencias para mejorar la interfaz de usuario:** un usuario considera la interfaz bastante completa y no tendría sugerencias. El otro sugiere la implementación de un apartado de "favoritos" donde los usuarios puedan guardar búsquedas concretas de su interés.

4.8.2. Resultados perfil técnico agrícola

En este test se han recibido dos respuestas, por un lado de Juan Diego Melero Blanca, gerente de Agroquímicos Calatrava y Manuel Jesús Carpio Arroyo, agricultor avanzado.

- **Relevancia y utilidad de la información presentada:** los usuarios coinciden en que la información es muy relevante y útil, especialmente en temas de humedad

y vegetación. Destacan la parte de enfermedades, que está bien resumida y proporciona datos cruciales, como los requisitos para la aparición de la mosca.

- **Opinión sobre el buscador de fincas mediante el código SigPac:** ambos consideran que el buscador de fincas mediante el código SigPac es eficaz, permitiendo identificar las fincas fácilmente y rápidamente a través del mapa.
- **Claridad y facilidad de interpretación de los datos y gráficos:** los usuarios encuentran que la presentación de los datos y gráficos es clara y fácil de interpretar. Mencionan que la información detallada aparece rápidamente al pulsar en cualquier apartado, facilitando la búsqueda de datos.
- **Facilidad de uso y navegación:** los usuarios consideran que la página web es fácil de usar y navegar, incluso sin experiencia técnica previa. Aseguran que cualquier persona ligeramente familiarizada con los temas encontrará la navegación sencilla.
- **Sugerencias para mejorar la interfaz de usuario:** un usuario sugiere incluir datos sobre el año hidrológico, que va de octubre a septiembre del año siguiente, ya que es una forma común de referirse a los datos de lluvia en el ámbito agrícola. Además, propone la opción de elegir varios puntos en una parcela para obtener datos específicos de esos puntos, como la presencia de un pozo.

En este capítulo se ha desarrollado todo el sistema de forma detallada, para concluir el trabajo, nos quedarían explicar las conclusiones del mismo, junto con un estudio de mercado y el trabajo futuro que se podría realizar. Por último, se presentará el manual de usuario.

Capítulo 5

Conclusiones y trabajos futuros

En este apartado, se explicarán las diferentes conclusiones a las que se llegaron tras la realización del proyecto, un estudio de posible comercialización y, por último, posible trabajo futuro que se podría realizar.

5.1. Estudio de comercialización

En este capítulo se incluye, en primer lugar, un estudio sobre el posible impacto que podría tener la aplicación en el mercado, en primer lugar, se presentarán los objetivos de este estudio, luego se analizará el entorno y finalmente se planteará el plan de mercadotecnia.

5.1.1. Objetivos

Los objetivos que podemos plantear a la hora de introducir nuestra aplicación en el mercado son los siguientes:

- **Competitividad en el Mercado:** establecer una herramienta competitiva que ofrezca funcionalidades innovadoras, como se explicó en el apartado contexto del proyecto [1.2](#), no existe una aplicación con las funcionalidades que se plantean en este apartado.
- **Estandarización y Centralización:** se podría crear una plataforma única y centralizada para el análisis del olivar, en la que cada usuario podría introducir sus

fincas y poder monitorearlas.

- **Satisfacción de las Necesidades del Mercado Agrícola:** proporcionar una herramienta completa y efectiva que satisfaga las necesidades específicas de los técnicos agrícolas en la región de Jaén y otras áreas con cultivos de olivar.

5.1.2. Análisis del Entorno

Mercado Potencial: según la [Organización Colegial de la Ingeniería Técnica Agrícola](#), se estima que hay más de 15.000 técnicos colegiados. Además, también podemos contar con agricultores avanzados.

Demanda del Mercado Agrícola: existe una creciente demanda de herramientas tecnológicas que ayuden a los agricultores y técnicos agrícolas a optimizar la producción y salud de los cultivos, especialmente en regiones como Jaén, con cultivos especializados como el olivar.

Tecnología Disponible: actualmente, la mayoría de personas disponen de un ordenador o smartphone, por ello, la tecnología web y de bases de datos permite el desarrollo de aplicaciones avanzadas que facilitan el monitoreo y análisis de la salud de los cultivos de manera eficiente y precisa, donde la mayor parte de técnicos agrícolas podrían acceder sin problema.

Competencia: aunque existen algunas aplicaciones en el mercado que ofrecen funcionalidades similares, la mayoría carece de características específicas como como la visualización interactiva de índices de vegetación, datos climáticos y tablas de enfermedades específicas para el olivar.

5.1.3. Plan de Mercadotecnia

El plan de mercado consistiría en llevar a cabo las siguientes actividades:

Campaña de Lanzamiento En primer lugar, se debe llevar a cabo una campaña de lanzamiento para dar a conocer la aplicación entre los técnicos agrícolas y otras partes interesadas en el sector agrícola. La campaña incluirá anuncios en páginas de compra de agroquímicos, por ejemplo, o páginas que sean de uso frecuente de los técnicos, además de publicaciones en redes sociales y correos electrónicos dirigidos a asociaciones agrícolas y empresas del sector.

Demostraciones y Pruebas Gratuitas

Para los técnicos agrícolas interesados, se podría hacer una demostración de la aplicación web con algunas funcionalidades básicas al igual que proporcionar pruebas gratuitas de la aplicación en colaboración con asociaciones agrícolas, cooperativas y entidades gubernamentales.

Estas actividades permitirán obtener retroalimentación de los usuarios y validar la eficacia de la aplicación en condiciones reales.

Alianzas Estratégicas

Se establecerán alianzas estratégicas con empresas del sector agrícola, instituciones de investigación y otras organizaciones relacionadas. Estas alianzas ayudarán a promocionar la aplicación y aumentar su alcance en el mercado, además de proporcionar acceso a redes de contactos clave.

Marketing de Contenidos

Se creará y compartirá contenido relevante sobre el monitoreo de la salud del olivar y las mejores prácticas agrícolas a través de blogs, vídeos en plataformas como TikTok y Youtube y seminarios web. Este enfoque de marketing de contenidos ayudará a posicionar la aplicación en el sector y atraerá a potenciales usuarios interesados en mejorar sus prácticas agrícolas.

Medición y Optimización Por último, se realizará un seguimiento continuo del rendimiento de la aplicación, incluyendo métricas de descargas, uso y retroalimentación de los usuarios. En base a estos datos, se realizarán ajustes y optimizaciones en la estrategia de mercado para mejorar la efectividad de la aplicación.

5.2. Conclusiones

A través del desarrollo de este proyecto, se ha logrado proporcionar una aplicación web a los técnicos agrícolas, que sea una herramienta innovadora y práctica para el monitoreo continuo y detallado de la salud del olivar. A continuación se resumen las conclusiones alcanzadas en función de los objetivos establecidos:

■ Desarrollo e Implementación de la Aplicación Web:

Se ha logrado desarrollar una aplicación web robusta y funcional que permite un

análisis temporal de la salud del olivar. La interfaz es simple y fácil de usar, lo que facilita a los usuarios ver y analizar información sobre el estado de los olivares a lo largo del tiempo.

■ **Conexión Efectiva con la Base de Datos:**

La aplicación está conectada eficazmente a una base de datos de Oracle que almacena todos los datos para el análisis de la salud del olivar. Se ha asegurado la integridad y la disponibilidad de la información, permitiendo un acceso rápido y fiable a los datos necesarios para el análisis y visualización en la página web.

■ **Mapas Interactivos de Información:**

Se han implementado mapas interactivos que permiten a los usuarios interactuar con los datos geoespaciales de los olivares, tanto a nivel de finca como de olivo, mostrándose en diferentes partes de la Página Web según el caso.

■ **Visualización de Índices de Vegetación y Datos Climáticos:**

La incorporación de la visualización de índices de vegetación como NDVI, NDMI y SAVI en gráficas, junto con datos de temperaturas y precipitaciones, proporciona a los usuarios herramientas efectivas para evaluar la salud de los olivares en función de la vegetación y las condiciones climáticas.

■ **Tabla de Enfermedades:**

Se ha desarrollado una tabla detallada de enfermedades que afectan a los olivares. Estas tablas permiten a los usuarios acceder a información específica sobre las enfermedades más comunes, facilitando la identificación y el diagnóstico de problemas de salud en los cultivos.

En conclusión, la aplicación web creada satisface los objetivos al ofrecer una herramienta completa y efectiva para el monitoreo y análisis de la salud del olivar. Especialmente, en la región de Jaén, esta solución permite a los técnicos agrícolas anticipar y responder proactivamente a amenazas potenciales, optimizando la salud y productividad de sus cultivos de olivar.

5.3. Trabajo futuro

En esta sección se presentarán algunas ideas sobre las posibles mejoras y adiciones que podrían hacerse a la aplicación en el futuro para ampliar sus funcionalidades.

En primer lugar, se podría hacer la implementación de un sistema automático para la detección de enfermedades. Este sistema podría utilizar técnicas de inteligencia artificial y aprendizaje automático para analizar continuamente los datos recolectados y proporcionar avisos tempranos a los agricultores sobre la presencia de enfermedades. Esto permitiría una respuesta más rápida y efectiva, ayudando a minimizar los daños y a mantener la salud del olivar.

Para realizar lo anteriormente mencionado, se debería implementar un sistema de actualización semiautomática de datos, que permita la incorporación de nuevos datos de forma semanal para mejorar significativamente la precisión de los análisis.

Además, con una actualización constante de los factores sobre el estado de los olivares, se podrían prever con mayor exactitud las próximas cosechas, así como la aparición de enfermedades y otros problemas potenciales. Este enfoque proactivo permitiría a los agricultores planificar con antelación y tomar medidas preventivas adecuadas.

Otra posible mejora sería la inclusión de herramientas específicas para el tratamiento del olivar. Si se realizasen las mejoras explicadas anteriormente, estas herramientas podrían proporcionar recomendaciones sobre el uso de pesticidas, fertilizantes y otras prácticas agrícolas basadas en los datos actuales y las predicciones. Al ofrecer soluciones personalizadas y basadas en datos, se optimizaría el cuidado del olivar, mejorando tanto la productividad como la sostenibilidad de los cultivos.

En resumen, la implementación de detección automática de enfermedades, la actualización semiautomática de datos y la inclusión de herramientas de tratamiento representarían avances significativos en la funcionalidad de la aplicación. Estos desarrollos contribuirían a una gestión más eficiente y efectiva de los olivares, beneficiando a los agricultores mediante una mayor productividad y sostenibilidad.

Bibliografía

- [1] Unknown. Jaén y el olivo, 2019. URL <https://centrodeolivaryaceite.com/es/jaen-y-el-olivo/>. Comprobado en 2024-03-11.
- [2] Aplicación SIGPAC | Portal Web del FEGA. URL <https://www.fega.gob.es/es/pepac-2023-2027/sistemas-gestion-y-control/sigpac>.
- [3] Pesca y Alimentación Ministerio de Agricultura. Sistema de información geográfica de parcelas agrícolas (sigpac), 2024. URL <https://www.mapa.gob.es/es/agricultura/temas/sistema-de-informacion-geografica-de-parcelas-agricolas-sigpac/default.aspx>. Comprobado en 2024-03-11.
- [4] Instituto de Magnetismo Aplicado (IMA). URL <https://www.ucm.es/ima/campos-electromagneticos-y-efectos-biologicos>.
- [5] Scientific Committees. Espectro electromagnético. URL <https://ec.europa.eu/health/opinions/es/lamparas-bajo-consumo/glosario/def/espectro-electromagnetico.htm>. Comprobado en 2024-03-11.
- [6] fbustos@yc.cl. Qué son los índices de vegetación y cómo se aplican en la agricultura de precisión, May 2018. URL <https://acortar.link/ds9j7U>.
- [7] David Revelo Luna, Julio Mejía Manzano, Bibiana P. Montoya-Bonilla, Javier Hoyos García, David Revelo Luna, Julio Mejía Manzano, Bibiana P. Montoya-Bonilla, and Javier Hoyos García. Análisis de los índices de vegetación NDVI, GNDVI y NDRE para la caracterización del cultivo de café (*Coffea arabica*). *Ingeniería y Desarrollo*, 38(2):298–312, December 2020. ISSN 0122-3461. doi: 10.14482/inde.38.2.628. URL http://www.scielo.org.co/scielo.php?script=sci_abstract&pid=S0122-34612020000200298&lng=en&nrm=iso&tlng=es. Publisher: Fundación Universidad del Norte.
- [8] Oliver Gutiérrez-Hernández. Tendencias recientes del NDVI en Andalucía: los límites del reverdecimiento. September 2022. doi: 10.21138/bage.3246. URL <https://doi.org/10.21138/bage.3246>.

[//riuma.uma.es/xmlui/handle/10630/30447](http://riuma.uma.es/xmlui/handle/10630/30447). Accepted: 2024-02-14T11:57:44Z
Publisher: Asociación Española de Geografía (AGE).

- [9] Diego Jesús Rodríguez Mauricio. Análisis temporal del NDVI del humedal de Purumpampa en Huamachuco y su relación con la expansión urbana. *Revista Geográfica de América Central*, (70):428–447, June 2023. ISSN 2215-2563. doi: 10.15359/rgac.70-1.16. URL http://www.scielo.sa.cr/scielo.php?script=sci_abstract&pid=S2215-25632023000100428&lng=en&nrm=iso&tlng=es. Publisher: <https://creativecommons.org/licenses/by-nc-sa/4.0/>.
- [10] NDVI: Fórmula Y Uso Del Índice De Vegetacion Na Agricultura, September 2021. URL <https://eos.com/es/make-an-analysis/ndvi/>. Comprobado en 2024-05-28.
- [11] A. R Huete. A soil-adjusted vegetation index (SAVI). *Remote Sensing of Environment*, 25(3):295–309, August 1988. ISSN 0034-4257. doi: 10.1016/0034-4257(88)90106-X. URL <https://www.sciencedirect.com/science/article/pii/003442578890106X>.
- [12] RPubs - Calculo de índices de vegetación, . URL <https://rpubs.com/Paraguay/1006360>. Comprobado en 2024-05-28.
- [13] Oleksandra Strashok, Monika Ziemiańska, and Vitalii Strashok. Evaluation and Correlation of Sentinel-2 NDVI and NDMI in Kyiv (2017–2021). *Journal of Ecological Engineering*, Vol. 23(nr 9), 2022. ISSN 2299-8993. doi: 10.12911/22998993/151884. URL <http://yadda.icm.edu.pl/baztech/element/bwmeta1.element.baztech-c57218ac-b26b-4027-99b2-8bfdd8065a81>.
- [14] NDMI: Fórmula Del Índice E Interpretación De Valores, February 2022. URL <https://eos.com/es/make-an-analysis/ndmi/>. Comprobado en 2024-05-28.
- [15] Darius Phiri, Matamyo Simwanda, Serajis Salekin, Vincent R. Nyirenda, Yuji Murayama, and Manjula Ranagalage. Sentinel-2 Data for Land Cover/Use Mapping: A Review. *Remote Sensing*, 12(14):2291, January 2020. ISSN 2072-4292. doi: 10.3390/rs12142291. URL <https://www.mdpi.com/2072-4292/12/14/2291>. Number: 14 Publisher: Multidisciplinary Digital Publishing Institute.
- [16] MODIS Web, . URL <https://modis.gsfc.nasa.gov/>. Comprobado en 2024-05-28.
- [17] A. Benali, A. C. Carvalho, J. P. Nunes, N. Carvalhais, and A. Santos. Estimating air surface temperature in Portugal using MODIS LST data. *Remote Sensing of Environment*, 124:108–121, September 2012. ISSN 0034-4257. doi: 10.1016/

- j.rse.2012.04.024. URL <https://www.sciencedirect.com/science/article/pii/S0034425712002003>.
- [18] Joaquín Muñoz-Sabater, Emanuel Dutra, Anna Agustí-Panareda, Clément Albergel, Gabriele Arduini, Gianpaolo Balsamo, Souhail Boussetta, Margarita Choulga, Shaun Harrigan, Hans Hersbach, Brecht Martens, Diego G. Miralles, María Piles, Nemesio J. Rodríguez-Fernández, Ervin Zsoter, Carlo Buontempo, and Jean-Noël Thépaut. ERA5-Land: a state-of-the-art global reanalysis dataset for land applications. *Earth System Science Data*, 13(9):4349–4383, September 2021. ISSN 1866-3508. doi: 10.5194/essd-13-4349-2021. URL <https://essd.copernicus.org/articles/13/4349/2021/>. Publisher: Copernicus GmbH.
- [19] E. Rodríguez, C. Lozano, and M. Campos. Detection of predation on *Euzophera pingüis* (Lepidoptera: Pyralidae) using an enzyme-linked immunosorbent assay. *European Journal of Entomology*, 102(4):793–796, 2005. ISSN 1210-5759. doi: 10.14411/eje.2005.108.
- [20] J M Durán and M Alvarado. Contribución al conocimiento de *Euzophera pinguis* (Haworth, [1811]) (Lep.: Pyralidae), plaga del olivo.
- [21] G. Katsikogiannis, D. Kavroudakis, T. Tscheulin, and T. Kizos. Population Dynamics of the Olive Fly, *Bactrocera oleae* (Diptera: Tephritidae), Are Influenced by Different Climates, Seasons, and Pest Management. *Sustainability (Switzerland)*, 15(19), 2023. ISSN 2071-1050. doi: 10.3390/su151914466.
- [22] Determinación del umbral de tratamiento para la mosca del olivo (*Bactrocera oleae* Gmel., Díptera: Tephritidae) en olivar destinado a la producción de aceite. .
- [23] Junta de Andalucía. El repilo del olivo y del acebuche. 2010.
- [24] M. Kaplan and İ. Alaserhat. Determination of Distribution, Population Change, Infestation and Damage situation of Olive Moth, *Prays oleae* (Bernard) (Lepidoptera: Praydidae) Causing Damage in Olive Orchards. *Erwerbs-Obstbau*, 62(3): 301–307, 2020. ISSN 0014-0309. doi: 10.1007/s10341-020-00493-y.
- [25] A Montiel Bueno. Factores de regulación de las poblaciones de *Prays oleae* (Bern.).
- [26] F.J. López-Escudero, J. Romero, R. Bocanegra-Caro, and A. Santos-Rufo. Predicting the Risk of *Verticillium* Wilt in Olive Orchards Using Fuzzy Logic. *Agriculture (Switzerland)*, 13(11), 2023. ISSN 2077-0472. doi: 10.3390/agriculture13112136.

- [27] Edwin Bautista-Villegas. Metodologías ágiles XP y Scrum, empleadas para el desarrollo de páginas web, bajo MVC, con lenguaje PHP y framework Laravel. *Revista Amazonía Digital*, 1(1):e168–e168, January 2022. ISSN 2810-8701. doi: 10.55873/rad.v1i1.168. URL <https://revistas.unamad.edu.pe/index.php/rad/article/view/168>. Number: 1.
- [28] Yohn Daniel Amaya Balaguera. Guía metodológica ágil, para el desarrollo de aplicaciones móviles “AEGIS-MD”. *Revista de Investigaciones UNAD*, 14(1):97–113, December 2015. ISSN 2539-1887. doi: 10.22490/25391887.1348. URL <https://hemeroteca.unad.edu.co/index.php/revista-de-investigaciones-unad/article/view/1348>. Number: 1.
- [29] Download | Scrum Guides, . URL <https://scrumguides.org/download.html>. Comprobado en 2024-05-28.
- [30] Desarrollo iterativo e incremental, September 2008. URL <https://proyectosagiles.org/desarrollo-iterativo-incremental/>.
- [31] El Modelo COCOMO. URL <http://www.sc.ehu.es/jiwdocoj/mmis/cocomo.htm>. Comprobado en 2024-06-03.
- [32] Agencia Tributaria: 1. Por coeficientes de amortización lineal, . URL <https://acortar.link/CyyODF>. Comprobado en 2024-06-03.
- [33] Justicia y Relaciones con las Cortes Ministerio de la Presidencia. XVIII Convenio colectivo estatal de empresas de consultoría, tecnologías de la información y estudios de mercado y de la opinión pública, June 2023. URL [https://www.boe.es/eli/es/res/2023/07/13/\(5\)](https://www.boe.es/eli/es/res/2023/07/13/(5)). Comprobado en 2024-06-03.
- [34] Quienes Somos – IEEE Sección España, . URL <https://ieeespain.org/quienes-somos/>. Comprobado en 2024-05-28.
- [35] Raúl Monferrer Agut. Especificación de Requisitos Software según el estándar de IEEE 830.
- [36] Descubre las soluciones de Oracle Database para lograr la máxima eficiencia, . URL <https://www.oracle.com/es/database/technologies/>.
- [37] Gabriel Méndez González. *Aprende a Desarrollar con Spring Framework: 2ª Edición*. IT Campus Academy, 2016. ISBN 9781541157378. Google-Books-ID: 4BbfDQAAQBAJ.

- [38] Rod Johnson, Juergen Hoeller, Keith Donald, Colin Sampaleanu, Rob Harrop, Thomas Risberg, Alef Arendsen, Darren Davison, Dmitriy Kopylenko, Mark Pollack, et al. The spring framework-reference documentation. *interface*, 21:27, 2004.
- [39] ¿qué es java spring boot? introducción a spring boot | microsoft azure. URL <https://azure.microsoft.com/es-es/resources/cloud-computing-dictionary/what-is-java-spring-boot>.
- [40] Zoraida Mamani Rodríguez, Luz Del Pino Rodríguez, and Juan Carlos Gonzales Suarez. Arquitectura basada en microservicios y devops para una ingeniería de software continua. *Industrial Data*, 2020. ISSN 1560-9146. URL <https://www.redalyc.org/articulo.oa?id=81665362014>.
- [41] Spring | marco de desarrollo de la junta de andalucía, . URL <https://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/93>.
- [42] Vue.js, . URL <https://vuejs.org/>. Comprobado en 2024-06-06.

Capítulo 6

Apéndices

En este apartado se presentará, en primer lugar, la guía original del Trabajo Fin de Título y, por otro lado, el manual de la aplicación.

6.1. Guía original del Trabajo Fin de Título



UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

PROPUESTA DE TRABAJO DE FIN DE GRADO

GRADO EN:

Grado en Ingeniería Informática

ESPECIALIDAD:

Ingeniería del Software

MENCIÓN:

General

TÍTULO DEL TRABAJO:

Aplicación web para el análisis temporal de la salud del olivar

Idioma:

Castellano

DESCRIPCION CORTA DEL TFG:

Este proyecto tiene como objetivo la creación de un portal web para visualizar e interactuar con toda la información que revela la salud del olivar. Se tomarán de entrada los índices de vegetación provenientes de imágenes multiespectrales procedentes de satélite y/o vuelos de dron a lo largo de un período de tiempo y de variables meteorológicas como temperatura y humedad. Esta información se enfoca al estudio de un conjunto de parcelas de olivos y también se estudiarán las plantas a nivel individual tras realizar un proceso de segmentación.

Esta aplicación web permitirá obtener información sobre la evolución de los valores de un conjunto de variables asociadas a la salud del olivar.

DATOS DEL TRABAJO FIN DE GRADO:

Modalidad del TFG

Proyecto de Ingeniería

Tipo de TFG

- ☐ TFG General
☒ TFG Específico

TFG en equipo

☐ TFG en Equipo
(Debe juntarse memoria justificativa)

Número máximo de estudiantes

1

TUTOR DEL TRABAJO FIN DE GRADO:

Tutor/a

Nombre:

LIDIA ORTEGA ALVARADO

Departamento:

Informática

A. Conocimiento:

LENGUAJES Y SISTEMAS INFORMÁTICOS

☐ Este TFG tiene un segundo tutor

DATOS DEL ALUMNO ASIGNADO

Alumno/a asignado/a

Nombre:

Alba Gómez Liébana

DNI:

77646750S

Dirección:

C/ Puerta de Jaén, 13, 2º Torredonjimeno, Jaén, 23650

Teléfono:

640535502

Correo-E:

agl00108@red.ujaen.es

CONOCIMIENTOS/REQUISITOS PREVIOS RECOMENDADOS

Es conveniente, que el estudiante haya cursado las asignaturas de Desarrollo de Aplicaciones Web, Algoritmos Geométricos y Programación de aplicaciones gráficas.

OBJETIVOS DEL TFG:

- Selección de variables procedentes de distintas fuentes de datos asociadas a la salud del olivar y su relación con el rendimiento de las cosechas.
- Implementación de una base de datos con la información de dichas variables a lo largo del tiempo, tanto a nivel de parcela como a nivel de algún conjunto de olivos individuales.
- Análisis, diseño e implementación del prototipo de una aplicación web para la visualización y análisis de mapas y de las diferentes variables relacionadas con la salud del olivar.

METODOLOGÍA A DESARROLLAR:

- Revisión bibliográfica sobre las tecnologías más adecuadas para el desarrollo del portal web
- Estudio de las variables que más influyen en la salud del olivar (diversos índices de vegetación, o variables meteorológicas o enfermedades)
- Análisis y diseño de la aplicación web
- Implementación de un prototipo software con gran parte de la funcionalidad final de acuerdo a las especificaciones anteriores
- Pruebas
- Documentación siguiendo la norma UNE 157801:2017
- Manuales asociados al prototipo.
- Generación de la memoria del trabajo realizado.

DOCUMENTOS Y FORMATOS DE ENTREGA:

- Prototipo software
- Documentación en formato electrónico siguiendo la norma UNE 157801:2017 para la elaboración de proyectos de sistemas de información
- Presentación de la exposición pública

DOCUMENTOS ENTREGADOS:

Documentos entregados

- ☒ Solicitud de propuesta de TFG cumplimentada y firmada.

6.2. Manuales de usuario

Para garantizar una experiencia óptima para los usuarios al interactuar con nuestra aplicación, se ha elaborado un manual de usuario. Este manual proporciona una guía detallada sobre cómo acceder y utilizar todas las funcionalidades del sistema de manera efectiva.

El objetivo es asegurar que los usuarios puedan aprovechar al máximo las capacidades de la aplicación y emplearlas de manera correcta para satisfacer sus necesidades. Con este manual, buscamos ofrecer una experiencia fluida y satisfactoria desde el primer contacto con la aplicación.

En primer lugar, se va a mostrar un diagrama con los diferentes recorridos que podría tener la aplicación web y posteriormente, se explicará cada una de las vistas.

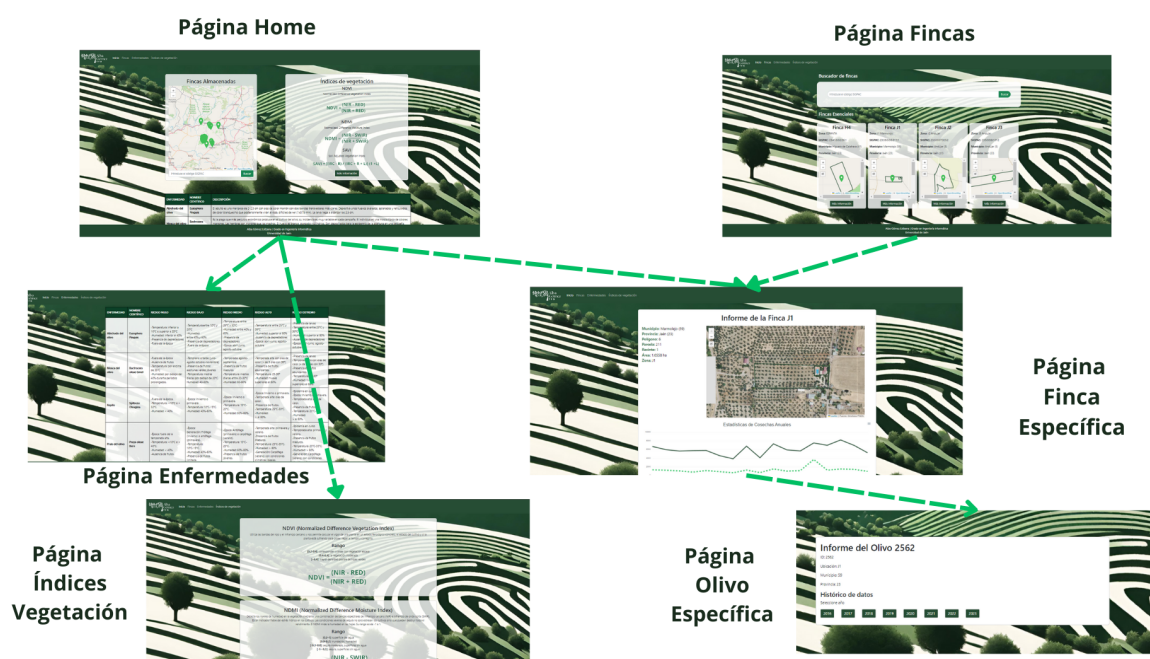


Figura 6.1: Diagrama de Recorridos de la Página Web - Elaboración Propia

En la Figura 6.1 vemos que desde la página inicial, denominada como Home, podemos ir a la página donde se explican las distintas enfermedades del olivar, a la página explicativa de los índices de vegetación y, por último, a la página específica de una finca en particular habiendo introducido su código SigPac.

Por otro lado, desde la página denominada Fincas, que se puede acceder desde la barra de navegación, podemos, de igual manera, acceder a la página específica de una finca y, si esta tiene olivos registrados, a la página específica de uno de ellos.

6.2.1. Página de inicio

En esta Página 6.2 se pretende tener una visión general de la página web, para que el usuario encuentre rápidamente lo que está buscando. En la Figura encontramos el resultado.

A la izquierda aparecería el mapa de las fincas que están registradas, si se conoce el código SigPac, se podría introducir en el buscador y acceder directamente a su página informativa 6.2.4.

A la derecha aparecen las distintas fórmulas de los índices de vegetación almacenados en la página, para un vistazo general, al pulsar al botón de más información, nos lleva a la página de los índices completa 6.2.3. Podemos también acceder a ella pulsando Índices de vegetación en la barra de navegación.

De igual manera, en la tabla situada en la parte inferior, aparecería un resumen de las enfermedades seleccionadas en el proyecto, si se pulsa en el botón de más información, nos dirigimos a la página donde aparece esta tabla completa 6.2.2. También podemos pulsar sobre Enfermedades en la barra de navegación.

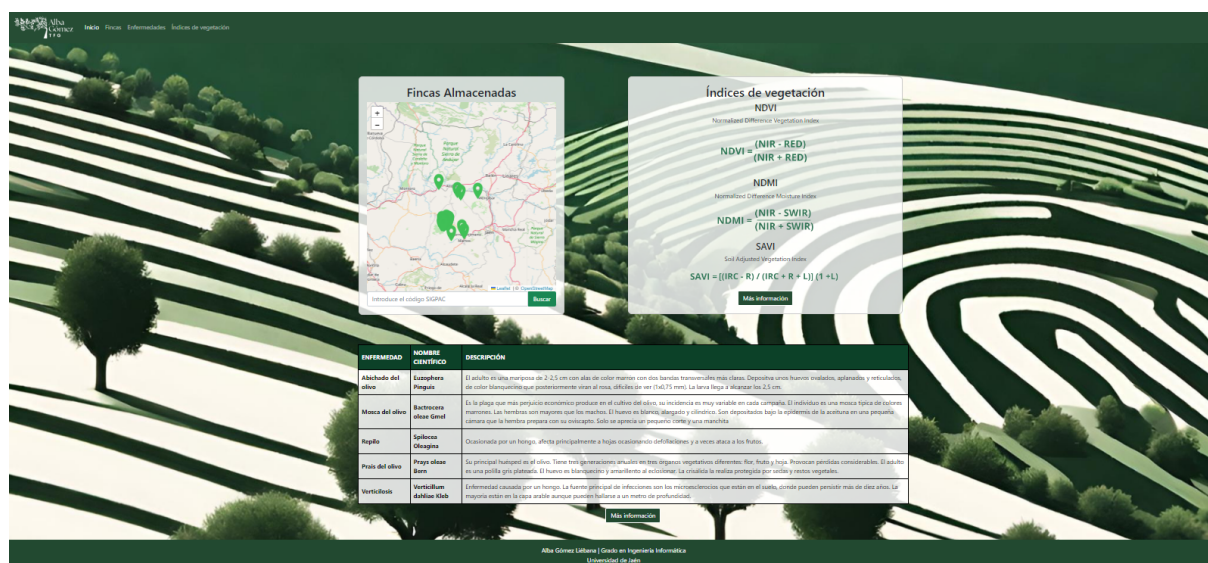
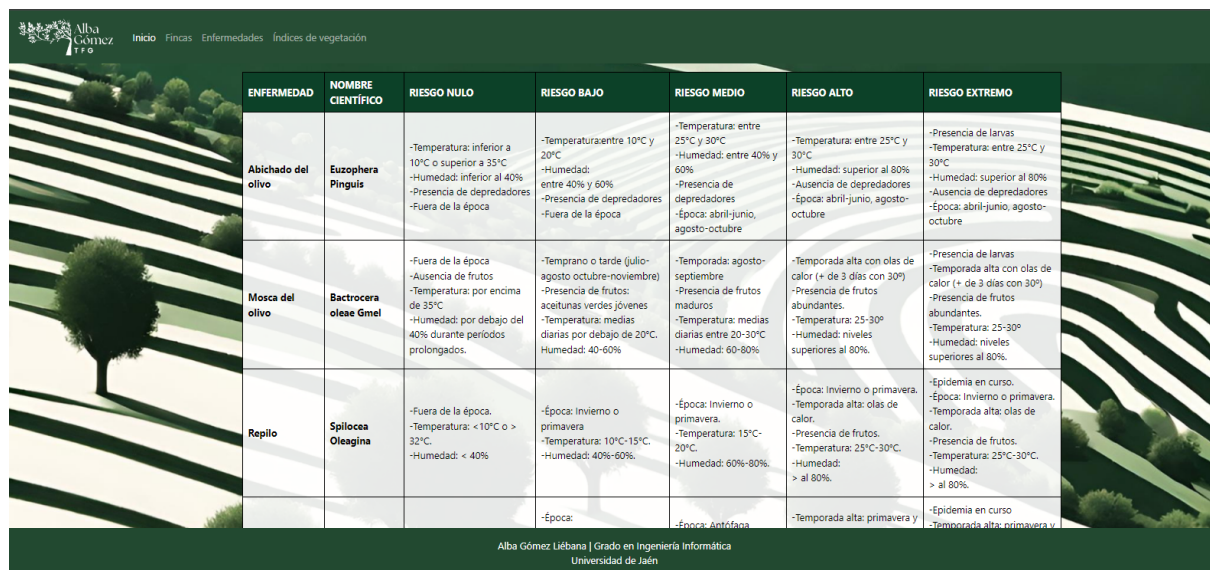


Figura 6.2: Página Home - Elaboración Propia

6.2.2. Página Enfermedades

En esta página podemos observar toda la información sobre las enfermedades del olivar, tenemos su nombre científico, común y una clasificación de los factores englobados en el rango de nulo a extremo. Esta página, junto con el análisis de las

gráficas, puede servir al usuario para diferenciar posibles enfermedades que estén ocurriendo en su olivar. En la Figura 6.3 tenemos como quedaría.



ENFERMEDAD	NOMBRE CIENTÍFICO	RIESGO NULO	RIESGO BAJO	RIESGO MEDIO	RIESGO ALTO	RIESGO EXTREMO
Abichado del olivo	Euzophera Pinguis	-Temperatura: inferior a 10°C o superior a 35°C -Humedad: inferior al 40% -Presencia de depredadores -Fuera de la época	-Temperatura: entre 10°C y 20°C -Humedad: entre 40% y 60% -Presencia de depredadores -Fuera de la época	-Temperatura: entre 25°C y 30°C -Humedad: entre 40% y 60% -Presencia de depredadores -Época: abril-junio, agosto-octubre	-Temperatura: entre 25°C y 30°C -Humedad: superior al 80% -Ausencia de depredadores -Época: abril-junio, agosto-octubre	-Presencia de larvas -Temperatura: entre 25°C y 30°C -Humedad: superior al 80% -Ausencia de depredadores -Época: abril-junio, agosto-octubre
Mosca del olivo	Bactrocera oleae Gmel	-Fuera de la época -Ausencia de frutos -Temperatura: por encima de 35°C -Humedad: por debajo del 40% durante periodos prolongados.	-Temprano o tarde (julio-agosto octubre-noviembre) -Presencia de frutos: aceitunas verdes jóvenes -Temperatura: medias diarias por debajo de 20°C. Humedad: 40-60%	-Temporada: agosto-septiembre -Presencia de frutos maduros -Temperatura: medias diarias entre 20-30°C -Humedad: 60-80%	-Temporada alta con olas de calor (+ de 3 días con 30°) -Presencia de frutos abundantes. -Temperatura: 25-30° -Humedad: niveles superiores al 80%.	-Presencia de larvas -Temporada alta con olas de calor (+ de 3 días con 30°) -Presencia de frutos abundantes. -Temperatura: 25-30° -Humedad: niveles superiores al 80%.
Repilo	Spilocca Oleagina	-Fuera de la época. -Temperatura: < 10°C o > 32°C. -Humedad: < 40%	-Época: Invierno o primavera -Temperatura: 10°C-15°C. -Humedad: 40%-60%.	-Época: Invierno o primavera. -Temperatura: 15°C-20°C. -Humedad: 60%-80%.	-Época: Invierno o primavera. -Temporada alta: olas de calor. -Presencia de frutos. -Temperatura: 25°C-30°C. -Humedad: > al 80%.	-Épidemia en curso. -Época: Invierno o primavera. -Temporada alta: olas de calor. -Presencia de frutos. -Temperatura: 25°C-30°C. -Humedad: > al 80%.
			-Época:	-Época: Antofagia	-Temporada alta: primavera y	-Épidemia en curso -Temporada alta: primavera y

Alba Gómez Liébana | Grado en Ingeniería Informática
Universidad de Jaén

Figura 6.3: Página Enfermedades - Elaboración Propia

6.2.3. Página Índices de Vegetación

En esta página tenemos información sobre los índices de vegetación que se muestran en las gráficas de la finca y del olivo. Para cada uno de los índices se muestra su abreviatura, su nombre completo, sus características, el rango de valores con su significado y, por último, la fórmula para su cálculo. En la Figura 6.4 se encuentra su aspecto.

6.2.4. Página Finca Específica

Una vez buscado o accedido a una finca correcta se mostrará una pantalla como podemos ver en la Figura 6.5, a la izquierda aparecerán todos sus datos y a la derecha un mapa con el contorno de la finca y, si tiene sus olivos segmentados en la BD, al pulsar sobre alguno de ellos, nos mandará a la Página 6.2.5.

Abajo, nos aparece un gráfico con los datos de las diferentes cosechas, si queremos obtener el de un año en particular, basta con poner el cursor encima suya y aparecerán tanto los kilogramos de aceite producidos como los kilogramos de aceituna recogidos.

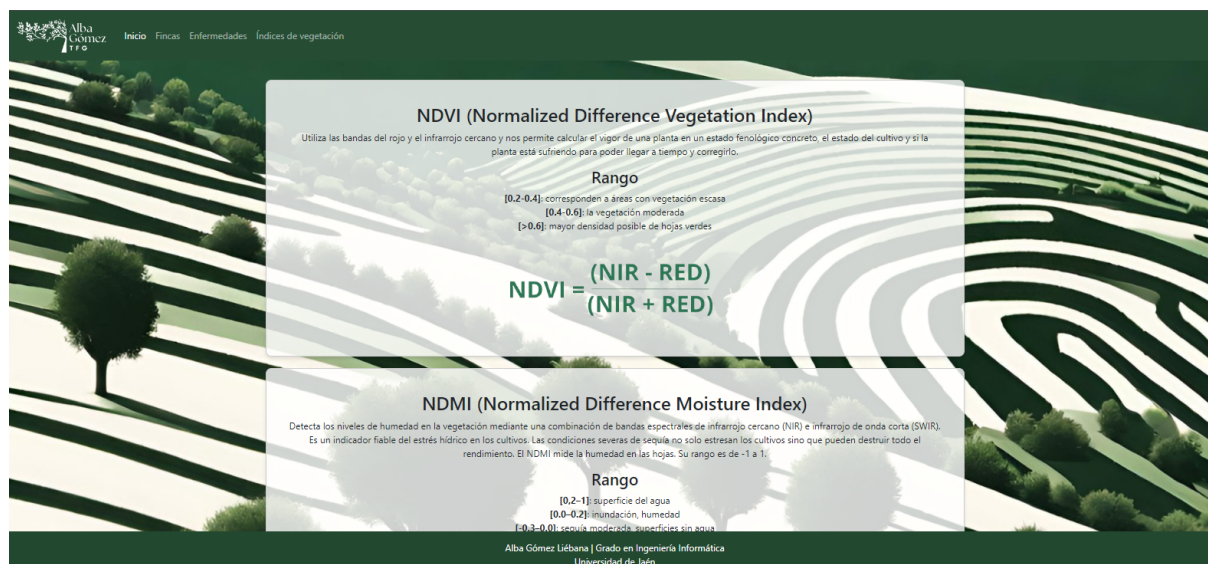


Figura 6.4: Página Índices de Vegetación de la Aplicación Web - Elaboración Propia

Si tenemos histórico de la finca registrado, aparecerán distintos botones con los años desde 2016 hasta 2023, si pulsamos sobre cualquiera de ellos y luego sobre Cargar Datos, nos aparecerán las distintas gráficas.

En primer lugar, aparecerían las tres sobre los índices de vegetación NDMI, NDVI y SAVI, luego la de las precipitaciones y, por último, la de las temperaturas. Las gráficas muestran la media por mes. Si en algún mes no hubiera datos, aparecería en blanco. De nuevo, si se quieren ver los datos más precisos, habría que poner el cursor encima del mes correspondiente. Si hay datos de dron cargados, aparecerán de color verde claro y los satelitales, provenientes de Sentinel-2 de verde oscuro.

6.2.5. Página Olivo Específico

Por último, tenemos la Página de olivo específico 6.6, en ella se muestra la información del olivo que hemos pulsado y tras ella, al igual que para la finca, una serie de botones representando los años del histórico, si se pulsa sobre cualquiera de ellos y se le da a Cargar Datos, aparecerán todas las gráficas sobre los índices de vegetación en específico para ese olivo. Si hay datos de dron cargados, aparecerán de color verde claro y los satelitales, provenientes de Sentinel-2 de verde oscuro.

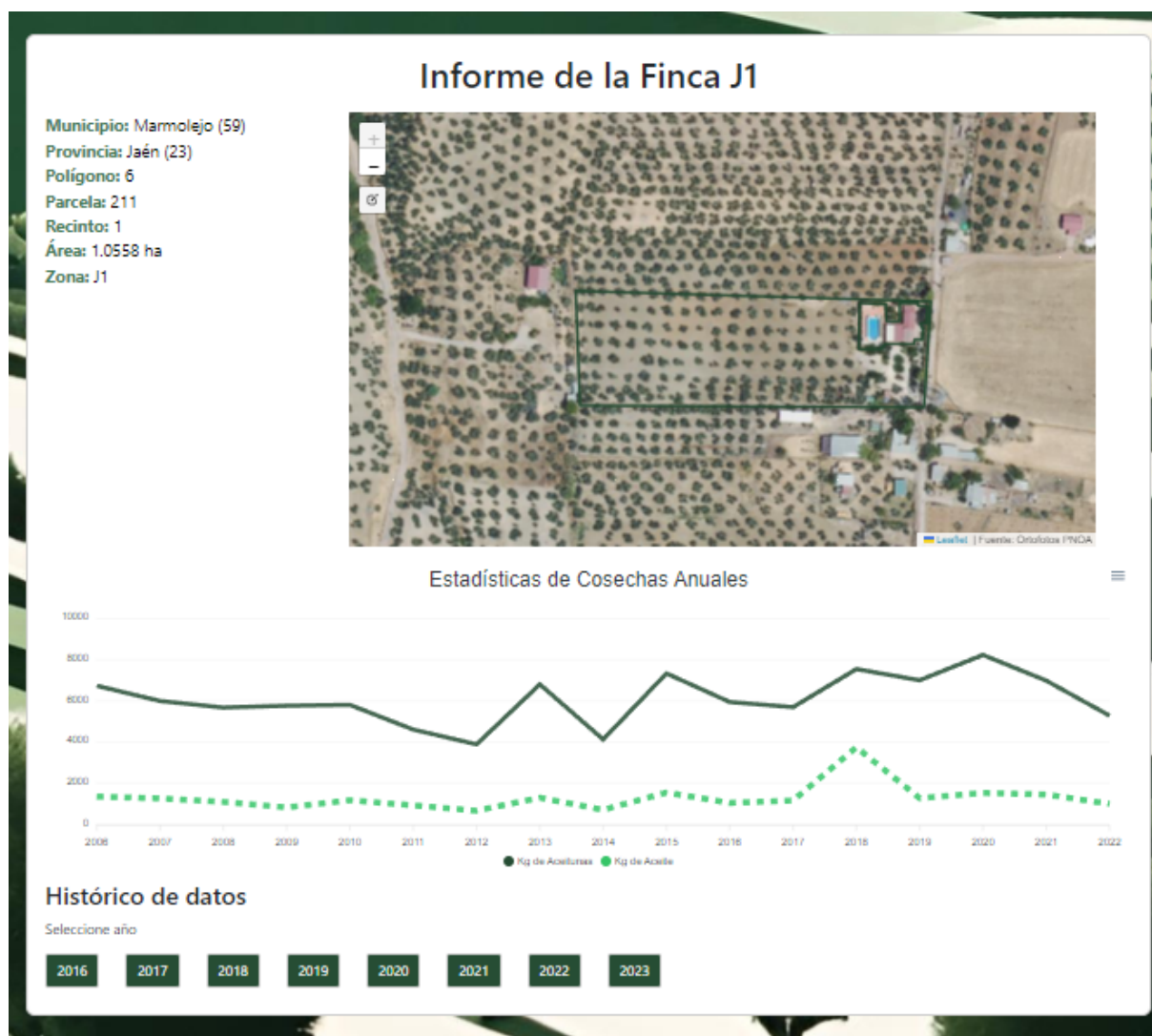


Figura 6.5: Página Finca Específica - Elaboración Propia

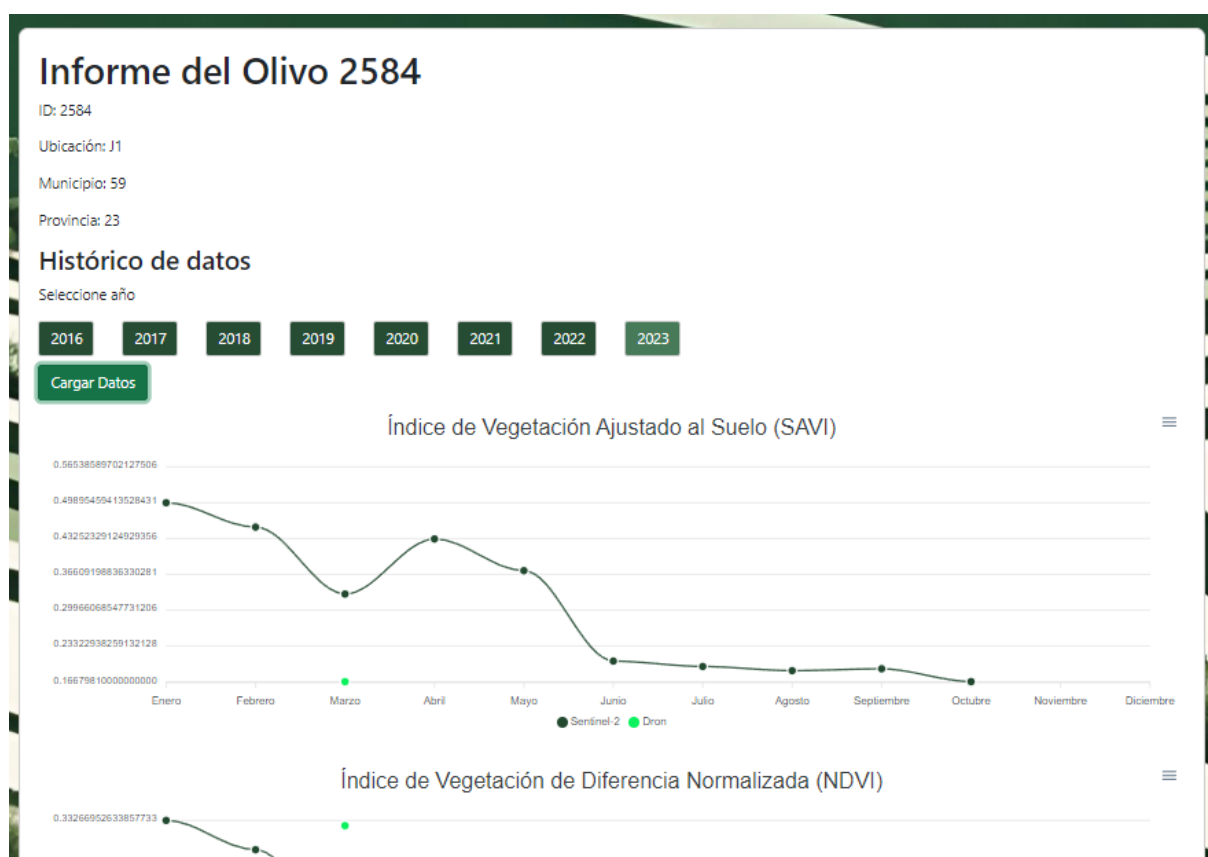


Figura 6.6: Página Olivo Específico - Elaboración Propia

