Universidad Zaragoza
1542

Trabajo Fin de Grado

Primer prototipo de gemelo digital de una explotación agraria especializada en frutos secos de cáscara

First prototype of a digital twin of a farm specializing in nuts and dried fruits

Autor

Jorge Laguna Argüello

Director

Javier Lacasta Miguel

Escuela de Ingeniería y Arquitectura

Ingeniería del Software

2021/2022

TÍTULO DEL PROYECTO

RESUMEN

La agricultura es “el arte de cultivar la tierra” (sadsma, s.f.) y nos provee la capacidad de obtener alimentos. Actualmente, la tecnología y las máquinas nos ayudan a tener una agricultura más eficiente y eficaz. El objetivo de este trabajo de fin de grado es mostrar las carencias de los sistemas actuales de gestión agrícola e investigar sobre cómo aplicar soluciones de Big Data y *machine learning* al mundo agropecuario, diseñando una arquitectura de sistema adaptado a nuestro tiempo. Específicamente, nos centramos en el uso de los llamados gemelos digitales, los cuales van más allá de la modelización y simulación tradicional de activos físicos, e introduce mejoras como simulaciones en tiempo real, mejora del diseño del producto y optimizaciones de productos y procesos del mundo real (¿Cuál es la diferencia entre una simulación y un gemelo digital?, 2020).

El presente TFG se ha enfocado en las explotaciones agrarias de frutos secos de cáscara, por su importancia frente a otros tipos de frutos (El cultivo de frutos secos, una apuesta rentable, 2016). En esta memoria, veremos los problemas a los que nos enfrentamos al aplicar un concepto tan nuevo y, en cierta parte, complicado de entender en su totalidad, sobre un campo tan poco explorado debido a la complejidad de modelar algo tan complicado como lo es un campo de cultivo: meteorología cambiante, uso de distintos fertilizantes o fitosanitarios, plagas impredecibles con daños impredecibles, tipo de planta con sus variedades, tipo de suelo y muchas otros pequeños detalles que pueden ser claves para simular su comportamiento. De la misma forma, veremos la complejidad que puede llegar a tomar una arquitectura de este estilo, proporcionando un primer prototipo que pueda servir como una primera simple aproximación del sistema final que se quiere diseñar.

Las tecnologías y las arquitecturas que se han escogido para hacer este primer prototipo del sistema son actuales, escalables y mantenibles en el tiempo. Veremos la aplicación web diseñada, cómo interactúan todos los componentes del sistema y, esencialmente, la infraestructura pensada para sostener el sistema en un futuro.

ABSTRACT

Agriculture is "the art of cultivating the land" (sadsma, s.f.) and provides us with the ability to obtain food. Currently, technology and machines help us to have a more efficient and effective agriculture. The objective of this final degree work is to show the shortcomings of current agricultural management systems and to investigate how to apply Big Data and machine learning solutions to the agricultural world, designing a system architecture adapted to our time. Specifically, we focus on the use of the so-called digital twins, which go beyond the traditional modeling and simulation of physical assets, and introduce improvements such as real-time simulations, product design improvement and optimizations of real-world products and processes (¿Cuál es la diferencia entre una simulación y un gemelo digital?, 2020).

The present thesis has focused on nut farms, due to their importance compared to other types of nuts (El cultivo de frutos secos, una apuesta rentable, 2016). In this report, we will see the problems we face when applying such a new concept and, to some extent, complicated to understand in its entirety, on a field so little explored due to the complexity of modeling something as complicated as a crop field: changing weather, use of different fertilizers or phytosanitary products, unpredictable pests with unpredictable damage, type of plant with its varieties, soil type and many other small details that can be key to simulate their behavior. In the same way, we will see the complexity that an architecture of this style can take, providing a first prototype that can serve as a first simple approximation of the final system to be designed.

The technologies and architectures that have been chosen to make this first prototype of the system are current, scalable and maintainable over time. We will see the designed web application, how all the components of the system interact and, essentially, the infrastructure designed to support the system in the future.

DECLARACIÓN DE AUTORÍA

DECLARACIÓN DE RESPONSABILIDAD Y AUTORÍA DEL TFG:

D.: Jorge Laguna Argüello, con DNI 73025588J, estudiante del Grado de Ingeniería informática de la Universidad de Zaragoza y tutelado por D.: Javier Lacasta Miguel

DECLARO QUE:

El Trabajo Fin de Grado denominado “Primer prototipo de gemelo digital de una explotación agraria especializada en frutos secos de cáscara”ha sido desarrollado respetando la propiedad intelectual (citando las fuentes bibliográficas utilizadas en la redacción de dicho trabajo), así como cualquier otro derecho, por ejemplo el de imagen que pudiese estar sujeto a protección del copyright.

En virtud de esta declaración afirmo que este trabajo es inédito y de mi autoría, por lo que me responsabilizo del contenido, veracidad y alcance del Trabajo Fin de Grado, y asumo las consecuencias administrativas y jurídicas que se deriven en caso de incumplimiento de esta declaración.

Y para que así conste, firmo la presente declaración en Zaragoza, a 23 de Noviembre de 2022.

Fdo.:

LAGUNA ARGÜELLO JORGE - 73025588J

Digitally signed by LAGUNA ARGÜELLO JORGE - 73025588J
Date: 2022.11.25 15:04:09 +01'00'

Este documento formará parte de la memoria del TFG correspondiente.

ÍNDICE

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

Esta página ha sido intencionalmente dejada en blanco

AGRADECIMIENTOS

En primer lugar, quiero agradecer de forma sincera la ayuda, la comprensión y la familiaridad de mis profesores Francisco Javier Zarazaga y Javier Lacasta Miguel, los cuales han pasado los últimos meses haciendo una gran labor en la supervisión del presente trabajo. Han puesto un gran interés en mí y en mis capacidades. También, mi gratitud hacia todos los profesores de esta carrera, por enseñarnos de la mejor forma posible los conceptos de esta rama de la tecnología tan maravillosa.

En segundo lugar, quiero dar las gracias a mis amigos de universidad, los cuales me han acompañado durante este largo, pero fructífero viaje por el mundo de la informática. Junto a ellos, ha sido posible conseguir un mayor entendimiento de las materias dadas y, por lo tanto, una mejora en mis capacidades. En especial, un fuerte abrazo a mis compañeros Noelia Oliete e Íñigo Aréjula.

De la misma forma, un especial agradecimiento a mis padres y a mi hermano. Los primeros por hacer posible mi entrada en la universidad y el acceso a mis estudios, gracias a su trabajo. Junto a ellos, mi hermano, de los cuales he recibido un cariño y un apoyo incondicional en las malas y en las buenas ocasiones. Como mención especial, también agradezco enormemente a mi novia por el cariño, el apoyo y los grandes momentos juntos, que han hecho que mi humor y mi felicidad hayan aumentado y hayan evitado momentos de frustración que, de otra forma, hubieran sido mucho peores.

Por último pero no menos importante, quiero agradecer el apoyo, las risas y los buenos momentos con mis amigos más próximos, que han ayudado a hacer estos cuatro años más llevaderos.

1 INTRODUCCIÓN

1.1 Contexto del trabajo

La agricultura tradicional ha estado con nosotros desde hace miles de años, cuando todo lo que se hacía en el campo estaba en manos de los seres humanos. Luego, fueron los animales los que ayudaron en las tareas más duras. Y más tarde, llegaron las máquinas que, comandadas por humanos, facilitaban enormemente las duras jornadas en las parcelas. Ahora, el paso natural es dejar gran parte de dicho trabajo a las máquinas, de forma automatizada, siempre al cargo de los seres humanos y con la ayuda de un sistema inteligente y autodidacta que ayude a la toma de decisiones, aumentando la eficiencia y disminuyendo los costes.

Junto a esto, el número de seres humanos en nuestro planeta ha aumentado de poco más de 6 mil millones a casi 8 mil millones en 2022 (Población mundial actual, s.f.), la aparición del capitalismo y el auge del poder adquisitivo de las clases medias han ocasionado un gran aumento, entre otras cosas, del consumo de alimentos en todo el mundo. Buena parte de la tierra fértil se dedica para el cultivo de todo tipo de plantas de forma masiva, que dan alimento a la creciente demanda de personas. Se necesitan de forma urgente nuevas soluciones adaptadas a la nueva época de modernización que se vive: una solución potente, escalable y mantenible en el tiempo.

En este momento, los sistemas que se acercan más a las llamadas soluciones de *smart/precision farming* son los sistemas Farm Management Information Systems (FMIS)[1], los cuales llevan unos años en el mercado, intentando dar soluciones (Kaloxylos, y otros, 2012). A pesar de ello, cuentan con varios problemas que se van a atacar en este TFG y cómo se pueden llegar a solventar con el sistema diseñado.

Junto a esto, Internet, la web, la inteligencia artificial y el Big Data[2] se han convertido en las plataformas perfectas para diseñar una aplicación que dé soluciones a los problemas de la agricultura descritos, superando las soluciones existentes a través de la ayuda y uso de los nuevos paradigmas en la ingeniería del software. Estos nuevos paradigmas incluyen el uso de protocolos web más eficientes, soluciones más escalables y mantenibles a nivel arquitectural, pero también a nivel de código, permitiendo dar una larga vida a un proyecto de software como este (Rathore, Shah, & Shukla, 2021).

[1] FMIS son sistemas usados para recolectar, procesar, guardar y diseminar datos en la forma que se requiera para llevar a cabo operaciones y funcionalidades relacionadas con las granjas.

[2] El big data (datos masivos) es el término que describe un gran volumen de datos, el cual crece de manera exponencial con el paso del tiempo (Big data: definición, tipos, características y beneficios).

1.2 Contexto tecnológico y herramientas utilizadas

Ya que las nuevas tecnologías y, sobre todo, la comunidad tan amplia de programadores, ofrecen la posibilidad de realizar este tipo de proyectos, se debe aprovechar dicha oportunidad. El proyecto se aborda como la elaboración de una propuesta de arquitectura software de referencia y la construcción de un sistema de información que dé visibilidad a la misma y a sus posibilidades. El trabajo también incluye una primera aproximación a la integración de fuentes de datos externas. De este modo, nos encontramos ante un sistema en el que los elementos tecnológicos fundamentales a desarrollar son aplicaciones web, servicios web, y servicios de alojamiento de datos.

Como se hablará en los próximos capítulos, uno de los componentes más importantes del sistema es la integración de datos de sistemas de información externa, que será el encargado de alimentar el gemelo digital diseñado. Una primera aproximación podría ser realizar un sistema más bien sencillo, siguiendo un paradigma monolítico. Esto podría funcionar bien si fuera un proyecto pequeño, con funcionalidad y alcance limitado. Sin embargo, no se sabe cuánto puede llegar a crecer y escalar el proyecto, pensando incluso en poder realizar un producto a nivel industrial para ser usado a una escala moderada – alta.

Para ello, se tienen que dejar de lado los antiguos paradigmas, tanto en referencia a la arquitectura del sistema como de la propia metodología y técnicas a la hora de la implementación del código. Una idea sería realizar un sistema con microservicios[3] distribuidos, altamente escalables y usando computación *serverless*[4] y servicios en la nube (Rasheed, Sauerwein, & Gounteni, 2022). Evidentemente, en los últimos tiempos, es imposible girar la cabeza a tecnologías en la nube, y se debe estar preparado para poder desplegar el sistema en este entorno para ganar eficiencia, velocidad y seguridad. No obstante, dado que es un primer prototipo, en este TFG no se han abordado estas soluciones.

Existe un principio muy conocido en el mundo del software que es "*Keep it Simple, Stupid!"* (KISS)[5], que dice que se mantenga lo más simple posible. Por ello, se ha decidido realizar un sistema con un enfoque que se aleja del monolito tradicional, pero no necesita abrazar completamente el modelo súper escalable y descentralizado, porque, de momento, no es necesario.

Este TFG se centra principalmente en el desarrollo de la infraestructura detrás del sistema que ha diseñado, y cómo interactúan cada uno de los componentes para ofrecer

---

[3] Los microservicios son un tipo de arquitectura que permite diseñar aplicaciones cuyos elementos funcionan de forma independiente pero coordinada (¿Qué son y para qué sirven los microservicios?, 2018).

[4] Se trata de un modelo de desarrollo directamente en la nube que posibilita el diseño y la ejecución de aplicaciones sin que sea necesario gestionar servidores (¿Qué es la informática sin servidor?, 2017).

[5] https://www.interaction-design.org/literature/article/kiss-keep-it-simple-stupid-a-design-principle

una solución. Por esto, la mayoría de tecnologías de las que se va a hablar pertenecen a la parte del *backend*.

Para esta parte contamos con MongoDB para el almacenamiento de datos; Prefect (un orquestador de *workflows* de Python) para realizar *pipelines* ETL (*Extract Transform Load*); Apache Camel y Apache Flink para integración de datos y comunicación con otros sistemas; RabbitMQ para la comunicación de entre las instancias de gemelos digitales y el gestor global de datos; gin-gonic (framework de Go) como servidor de acceso a los recursos de los gemelos digitales; Redis como caché de la información de los gemelos digitales y, en este caso, almacenamiento de sesiones; Svelte, un framework de Javascript para realizar el cliente web; Nginx como proxy inverso para exponer la aplicación web al mundo.

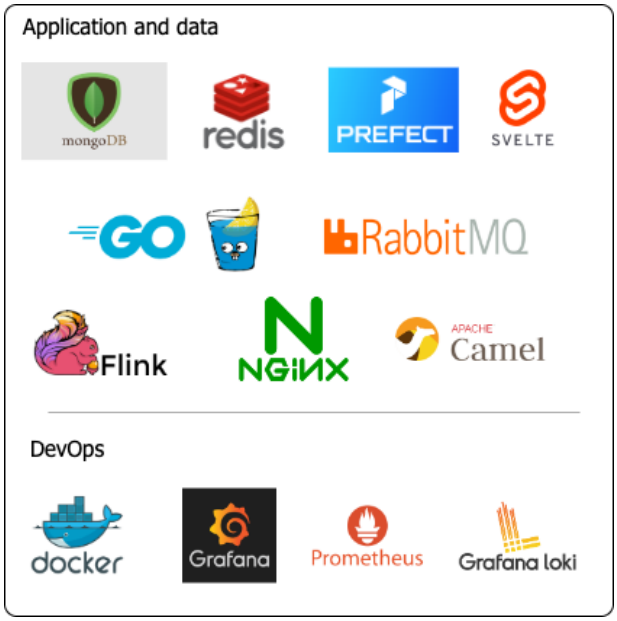


Ilustración 1: Stack de tecnologías usado en el TFG

Para la parte de DevOps, contamos con: Loki para centralizar la creación y recogida de logs; Prometheus para exportar métricas del sistema; Grafana para visualizar las métricas y los logs de forma cómoda; Docker para realizar el despliegue del sistema y hacerlo fácilmente en cualquier entorno.

1.3 Alcance, objetivos y limitaciones

Como hemos dicho en los puntos precedentes a este, existen varios factores por los cuales se considera necesario un paso hacia adelante en la modernización del campo. Entre estos factores se encuentran las subidas de los costes de producción, la poca atracción de la gente por la dura vida del campo o la masificación del mercado de la alimentación. De esta forma, los FMIS existentes tienen poca capacidad de adaptación a la nueva era de tecnologías y menos flexibilidad a la hora de escalar y el manejo de grandes cantidades de datos.

Los modelos de simulación tradicionales simplemente se limitan a recoger toda la información posible y realizar predicciones estadísticas, intentando inferir relaciones entre variables. El gemelo digital recoge grandes cantidades de datos para proporcionar un sistema de toma de decisiones, y realiza predicciones mucho más precisas gracias al aprendizaje automático gracias al manejo del Big Data. De esta forma, utilizando las tecnologías punta que existen y haciendo uso de su escalabilidad, fácil manejo de datos, asincronía de la web y de las buenas prácticas, conseguir un producto más redondo. En adición, otro de los objetivos (aunque en un futuro más lejano) sería la interoperabilidad del gemelo digital con otros FMIS o gemelos digitales, a través de un middleware que permita la comunicación entre los diferentes sistemas, realizando tareas conjuntas de análisis de datos y monitorización. Así, se consigue una estandarización de los sistemas y se convierte en un punto de acceso único para los clientes.

No obstante, el concepto de gemelo digital en agricultura está muy poco consolidado debido, especialmente, a la gran complejidad de discretizar una realidad basada en "elementos" vivos (una planta o conjunto de plantas, un animal, o conjunto de animales, una fruta o conjunto de frutas, una mezcla de todo lo anterior) y con unos ámbitos temporales (plantas con periodos de vida de un año frente a plantas con periodos de vida de cientos de años) y espaciales (desde una maceta o un pequeño corral, hasta conjuntos de explotaciones que pueden suponer la producción de una región o país) tan heterogéneos y complejos.

El presente TFG se enmarca en el proyecto GEDEFEC (GEmelo Digital para Explotaciones de Fruto sEcos de Cáscara, https://www.gedefec.es) que cuenta con el soporte económico parcial del programa de Apoyo a Agrupaciones Empresariales Innovadoras (AEI) del Ministerio de Industria, Comercio y Turismo. Este proyecto está siendo abordado por el grupo de Sistemas de Información Avanzados de la Universidad de Zaragoza, en colaboración con dos clústeres tecnológicos, cuatro PYME y otro centro de investigación. El proyecto aborda los siguientes retos:

- Fase I:
  - Modelización de la explotación: Identificación y caracterización de variables, Especificación de relaciones entre variables, y Especificación del modelo de datos del gemelo.
  - Orígenes de datos: Estado del arte en sensórica, Capacidades de adquisición de datos de la maquinaria, y Fuentes de datos abiertos en la Web.
- Fase II:
  - Automatización de datos de entrada y de datos de salida: Sensorización de insumos, Sensorización de laboreo, y Sensorización de agua.
  - Realimentación del gemelo en ciclo: Condicionantes de emisiones, Condicionantes de producción, y Condicionantes de lixiviados.
- Fase III:
  - Red de gemelos: Interacciones entre explotaciones, Algoritmia de interacción entre gemelos, y Funcionalidades vinculadas a la red.

- Inferencia de conocimiento: Identificación de relaciones y patrones subyacentes, Algoritmia de interacción entre gemelos, y Funcionalidades vinculadas a la inferencia de conocimiento.

Cada una de las fases lleva asociado el desarrollo de un prototipo de sistema de información que sirva como demostrador de los avances conceptuales y tecnológicos conseguidos en la misma. Este TFG supone el prototipo demostrador correspondiente a la primera de las fases.

1.4 Esquema general de la memoria del proyecto

La memoria de este proyecto se organiza de la siguiente forma. La sección 2 provee información sobre cómo son los gemelos digitales en relación a la agricultura, que es el objetivo del presente trabajo de fin de grado. Presenta una visión moderna de la arquitectura del sistema del gemelo digital a diseñar, haciendo hincapié en la importancia del Big Data y del análisis a través del *machine learning*. En adición a esto, se ve cómo se ha transformado dicha arquitectura conceptual en una arquitectura con las tecnologías introducidas en la sección 1.2, explicando cada parte de sistema en la distintas subsecciones, incluyendo la interfaz de usuario, el modelo de datos, la instancia del gemelo digital diseñado o el gestor de datos global de los diferentes gemelos digitales. Asimismo, se explica la dimensión del trabajo realizado y los problemas encontrados al realizarlo, los cuales son interesantes para tener en cuenta en futuras versiones.

En la sección 3 se explican las lecciones aprendidas y las conclusiones sobre el trabajo realizado, en el cual se ha aprendido a tener una visión más sofisticada de cómo debe ser un sistema real, escalable y mantenible, usando un buen número de herramientas. Además, se explican los proyectos futuros y funcionalidades próximas para añadir al prototipo.

En los anexos I y II tenemos una explicación más concreta sobre el contexto del trabajo, detallando la situación de la agricultura y cuál es el rol de los gemelos digitales en esta especialidad. Junto a esto, se especifica la relevancia de los frutos secos de cáscara para el trabajo y por qué se ha escogido para desarrollar un primer prototipo del gemelo digital.

En anexo III es un vistazo más detallado sobre el prototipo realizado, donde se incluyen: más información sobre las tecnologías usadas y no usadas, el listado completo de requisitos, guía de estilo de la aplicación web y la arquitectura de los componentes usados (*clean architectures*), entre otros.

2 GEMELOS DIGITALES EN LA AGRICULTURA

Para conseguir los objetivos, hemos propuesto varias funcionalidades principales. Queremos proporcionar al usuario una potente herramienta que permita predecir de forma efectiva los momentos críticos del agricultor: en qué momentos debería regar para reducir costes; cuál es el mejor momento para cultivar o cosechar las plantas; cuál es la cantidad efectiva y suficiente de fertilizantes y cada cuánto tiempo ponerlos; ser capaz de detectar irregularidades o amenazas en los datos recogidos y comunicarlos al usuario; ser capaz de monitorizar el terreno con la mayor frecuencia posible, proporcionando al usuario la mayor cantidad de datos útiles; permitir al usuario enviar comandos al gemelo digital para realizar acciones sobre el campo correspondiente; ser capaz de comunicarse con otros gemelos o FMIS para conseguir un mayor conocimiento, y por lo tanto, mayor capacidad de predicción, sobre lo que sucede en otros terrenos, diferentes o similares.

Con estas funcionalidades, se plantea la arquitectura del sistema que queremos construir y podemos observar cómo actúan los componentes del gemelo digital. En la Ilustración 2se puede visualizar un diagrama arquitectural a un nivel conceptual, centrado en la arquitectura Big Data. Tenemos, por un lado, el gestor global de datos y, por otro lado, los distintos gemelos digitales que se tengan. Cada gemelo corresponde a la modelización de un recinto de una parcela concreta y, para este TFG, serían recintos de frutos secos de cáscara.

El componente del gestor de datos global es, básicamente, el recolector de toda la información posible para alimentar a los distintos gemelos digitales con datos útiles. Aquí se recogen datos de distintas fuentes y sistemas de datos externos, y se almacenan de forma indefinida los datos sin procesar, de forma que estén siempre disponibles sin depender de terceros. La integración de los datos puede ser en forma de paquetes (*batch*[6]) que se extraen de forma periódica o bajo demanda, o en forma de flujo continuo de datos (*streamming*[7]) para recibir datos en tiempo real o casi tiempo real. Además, se comunica con los sistemas externos para recibir información que quieran enviarnos (otros FMIS, por ejemplo) o enviar (comandos a la sensórica del recinto, por ejemplo). La información del *data lake* es revisada cada cierto tiempo para realizar análisis más superficiales sobre la información en crudo.

[6] El procesamiento en *batch* es el método que los ordenadores usan para completar trabajos con grandes volúmenes de datos de forma periódica (AWS).

[7] El procesamiento en *streamming* es una técnica de gestión de datos que involucra integrar datos en *stream* de forma continua para analizar, filtrar, transformar o mejorar los datos en tiempo real de forma rápida (Lawton). Evidentemente, consume más energía que el *batch*.

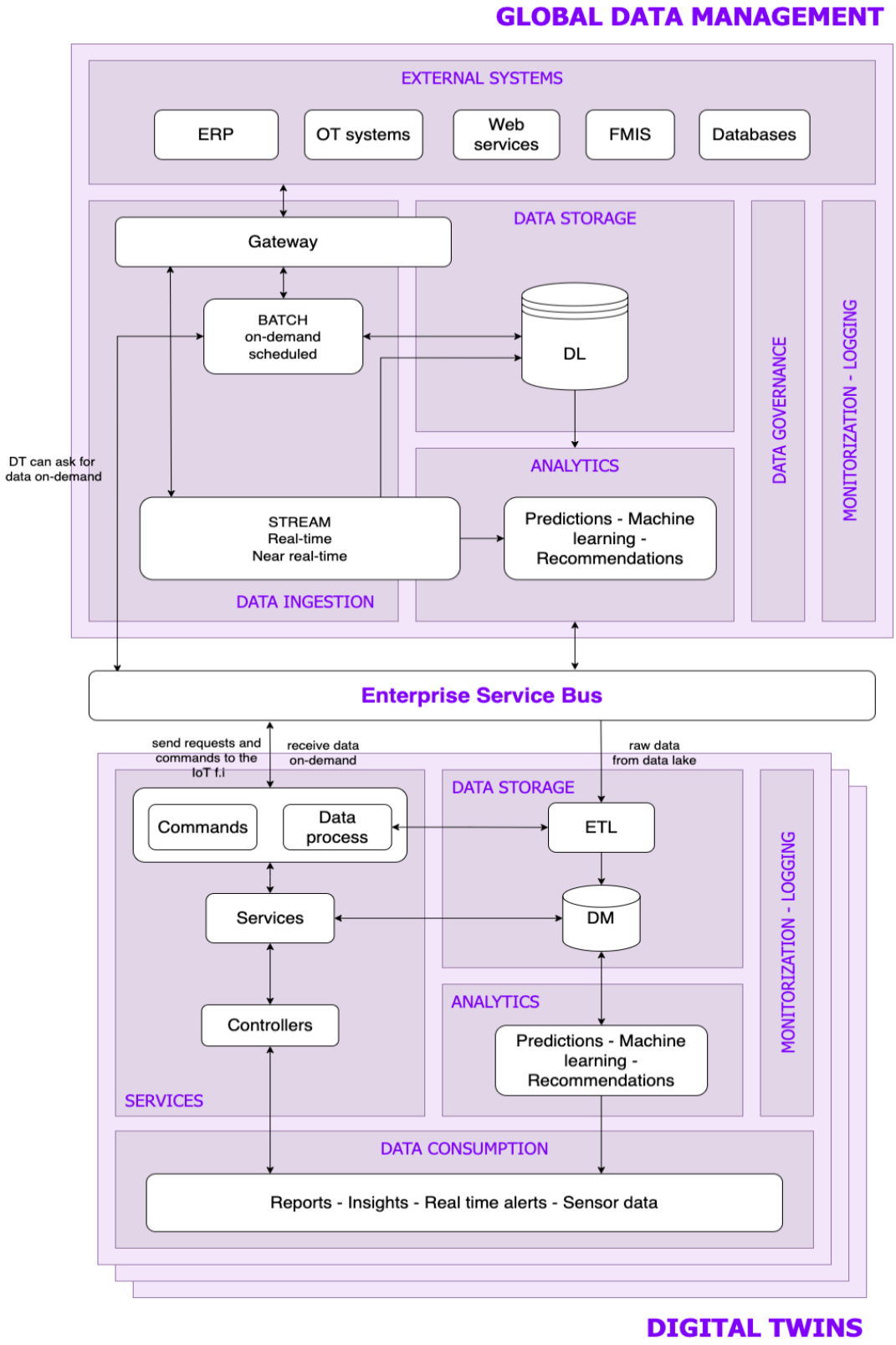


Ilustración 2: Diagrama de arquitectura conceptual. DM (data mart) y DL (data lake)

El componente del gemelo digital trata de tomar información específica de cada recinto del *data lake*, de forma que los datos son limpiados y procesados para que se puedan

analizar de una forma más acusada y concreta. La información es extraída de forma periódica para ser analizada, extraer predicciones, recomendaciones y realizar *machine learning*. De esta forma, generan reportes, alertas en tiempo real y otras formas de visualizar la información, que puede ser consumida por aplicaciones web como la que vamos a realizar.

Como vemos en la Ilustración 2, se observa también un Bus. Dicho bus comunica, tanto los distintos gemelos digitales con el gestor de datos global, como la comunicación entre los propios gemelos digitales, por si precisan saber, por ejemplo, cómo afecta un tratamiento con fitosanitarios de un recinto a otro recinto colindante. Esta comunicación puede realizarse tanto de forma síncrona como de forma asíncrona.

2.1 Propuesta arquitectural tecnológica

Una vez hemos visto el sistema a nivel conceptual, vamos a bajar al nivel tecnológico de la arquitectura. Hemos visto en la sección 1.2 las tecnologías que se han implementado en el sistema, de forma que ahora las veremos en el diagrama de la Ilustración 3.

Como primer prototipo de arquitectura, podemos ver que es algo más sencillo, alejado de lo que queremos conseguir como un producto apto para su distribución a nivel industrial. Sin embargo, podemos ver las partes principales que se han explicado anteriormente en la Ilustración 2: el componente del gemelo digital y el componente del gestor de datos global, además de la parte de monitorización del sistema. Como vemos, de momento, no existen servicios de predicción, *machine learning* o recomendación, dado que eso entra en una fase futura del proyecto.

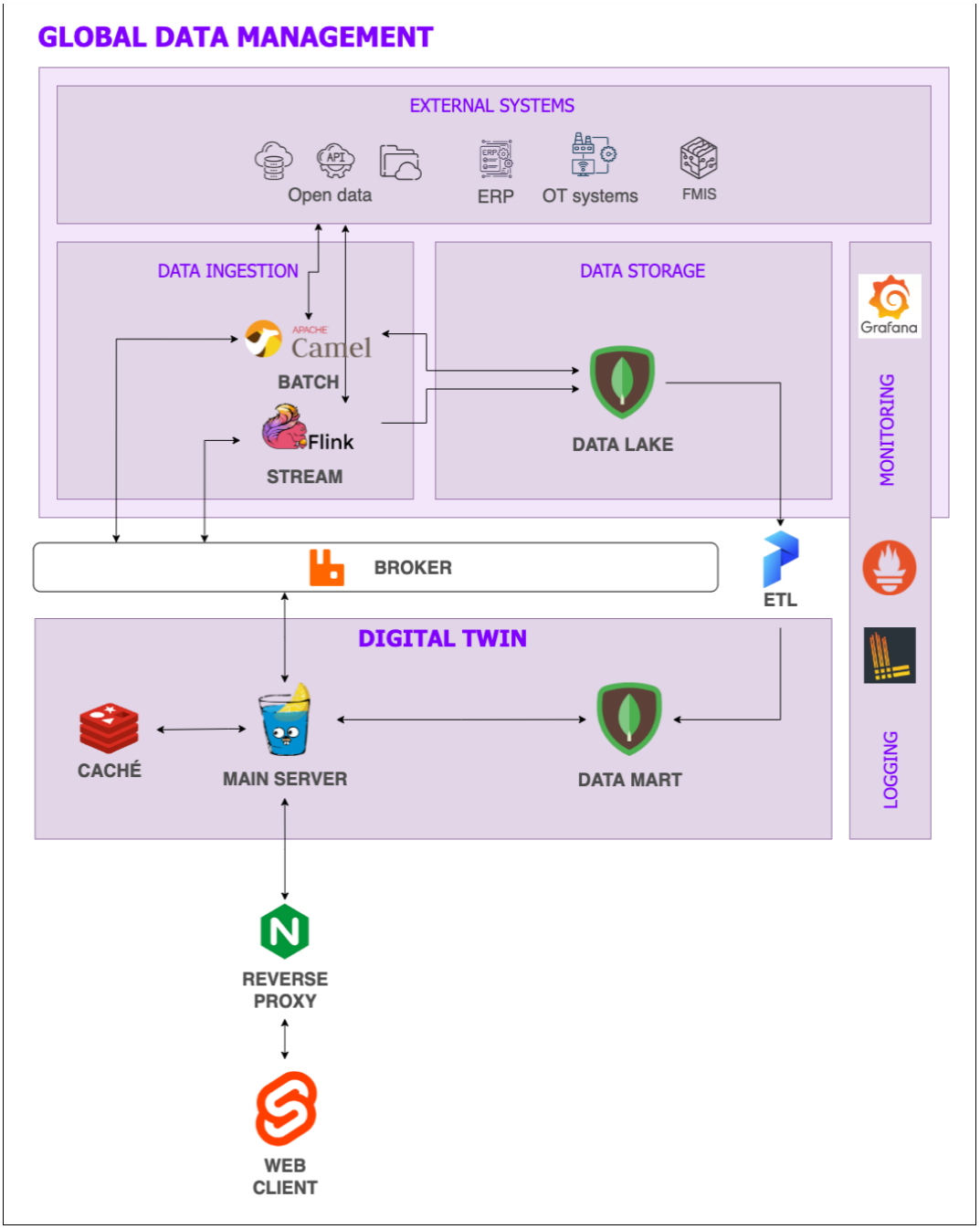


Ilustración 3: Diagrama de arquitectura a nivel tecnológico. Fase 1

En cuanto al despliegue de la aplicación, se hace de una forma sencilla. Todos los componentes que se ven en el diagrama de la Ilustración 3 se despliegan de forma sencilla en contenedores de Docker, a través de un archivo *docker-compose.yml*, sin contar con funciones más avanzadas como replicación de nodos. Estos contenedores se alojan, de momento, en una máquina del equipo (*self-hosted*), de forma que se expone a la web a través de una instancia de Nginx que existe en dicha máquina, permitiendo protegernos de ciertos ataques, actuando de caché y funcionando como *proxy* inverso. Asimismo, añade la capa de TLS a la comunicación HTTP aplicación a través de la herramienta Let's Encrypt, protegiendo aún más nuestras comunicaciones.

2.2 Desarrollo del prototipo de gemelo digital

Un campo que está en un invernadero tiene todas las variables que podrían afectar a la planta o fruta bajo control: el riego, la temperatura, la humedad, la cantidad de sol, los nutrientes, las plagas, entre otras cosas. Por ello, modelizar un campo así es casi trivial. No obstante, no ocurre lo mismo con los campos que se encuentran al aire libre. Como se ha contado en la sección 1.1, influyen muchas variables que pueden no controlarse, como la meteorología, plagas, agua o incluso otros campos cercanos.

Por esto, componer un dominio del problema robusto desde un principio es especialmente complicado. La metodología que se ha seguido para construir el sistema ha sido centrada en dos pilares: diseñar una interfaz para decidir cómo se iba a ver la parte de ayuda a la toma de decisiones del sistema con un panel de administración de las parcelas que observa el usuario; diseñar toda la infraestructura del sistema sin preocuparnos excesivamente del modelo de datos.

2.2.1 INTERFAZ DE USUARIO

Como no se tenía claro el dominio del problema, se empezó por hacer un prototipo de la interfaz de la aplicación web con Figma. Con este prototipo y los pocos datos reales de los que se disponía, se empezó a diseñar el modelo de datos que se podrían usar, por lo menos en una primera fase temprana del proyecto.

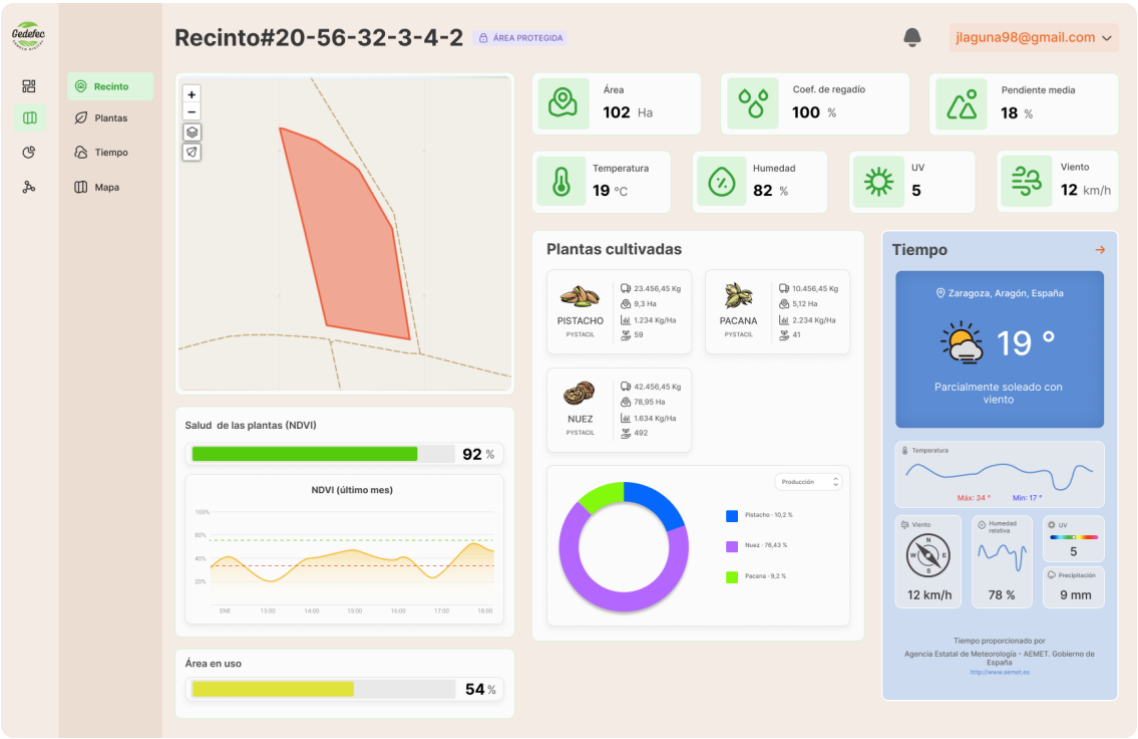


Ilustración 4: Panel de administración del gemelo digital. Se muestra la sección de un recinto individual, donde se ve un resumen de su estado actual. Fase 1

Se quería realizar una interfaz simple, con la información suficiente por sección, pero sin llegar a abrumar con demasiados componentes como gráficos o tablas. Dicha interfaz irá

aumentando su complejidad de forma progresiva cuando se implementen nuevas funcionalidades o se actualicen las que haya en ese momento. Se pueden visualizar todas las imágenes del prototipo en la sección 5 del Anexo III, además de una guía del estilo implementado.

2.2.2 MODELO DE DATOS

Una vez realizado el diseño, más o menos sabíamos el dominio que se estaba manejando más claramente, por lo que decidimos hacer un diseño más completo del modelo de datos que podemos ver en la Ilustración 5, el cual es agnóstico de cualquier tecnología que se utilice. Algunas entidades, las cuales se ven de la forma *[<...>]** han sido simplificadas por una mejor visualización del diagrama, pero se encuentran enumeradas en la nota de abajo a la derecha.

Tenemos claro que vamos a tratar con parcelas, las cuales son afectadas por la meteorología de la zona (diaria y pronóstico de varios días). Éstas se muestran con el estándar GeoJSON[8], de forma que se puedan realizar operaciones como saber si dos parcelas intersecan o a qué distancia se encuentran[9]. Cada parcela pertenece a una granja concreta, la cual posee maquinaria y herramientas, productos para diferentes propósitos, además de contar con asesores sanitarios y aplicadores (pesticidas). Asimismo, cada parcela está formada por una serie de recintos, los cuales también siguen el estándar GeoJSON. Éstos tienen características como el tipo de irrigación, el área usada, las plantas que hay en este momento, si forma parte de un área protegida, entre otros. También recibe información histórica de distintas fuentes: índices de NDVI[10] medios por parcela; información de riego; información de la tierra en la que se cultiva; las plantas que se cultivan en el recinto. Algunas de estas actividades hacen uso de la maquinaria, productos y otras cosas, como se ve en las relaciones de la Ilustración 5.

De la misma forma, los recintos, según vaya llegando la información, por ejemplo de los sensores, irán creando notificaciones para el usuario. El usuario de la aplicación, a su vez, podrá monitorizar una serie de parcelas, proporcionándose un resumen personalizado para cada usuario de todo lo que sea relevante (actividades, ganancias, pérdidas, avisos, recomendaciones, y otros). En cuanto a los recintos, el usuario podrá programar actividades (programar regadío, maquinaria), que quedan registradas. El usuario también puede realizar simulaciones bajo demanda del recinto que tenga disponible, produciéndose unos resultados que pueden ser añadidos al sistema de toma de decisiones diseñado.

---

[8] GeoJSON es una forma de estandarizar los objetos geométricos representados en JSON. Contemplan formas geométricas como puntos, líneas o polígonos. https://geojson.org

[9] MongoDB soporta operaciones sobre datos geoespaciales representados con GeoJSON. https://www.mongodb.com/docs/manual/geospatial-queries/

[10] Índice de vegetación de diferencia normalizada. Mide el verdor y la densidad de la vegetación captada en una imagen de satélite. https://eos.com/es/make-an-analysis/ndvi/

Así, la mayoría de datos se almacenan para su análisis, siendo la gran parte de ellos usados en series temporales en el sistema de ayuda a la toma de decisiones y para alimentar al modelo predictivo que se tenga.

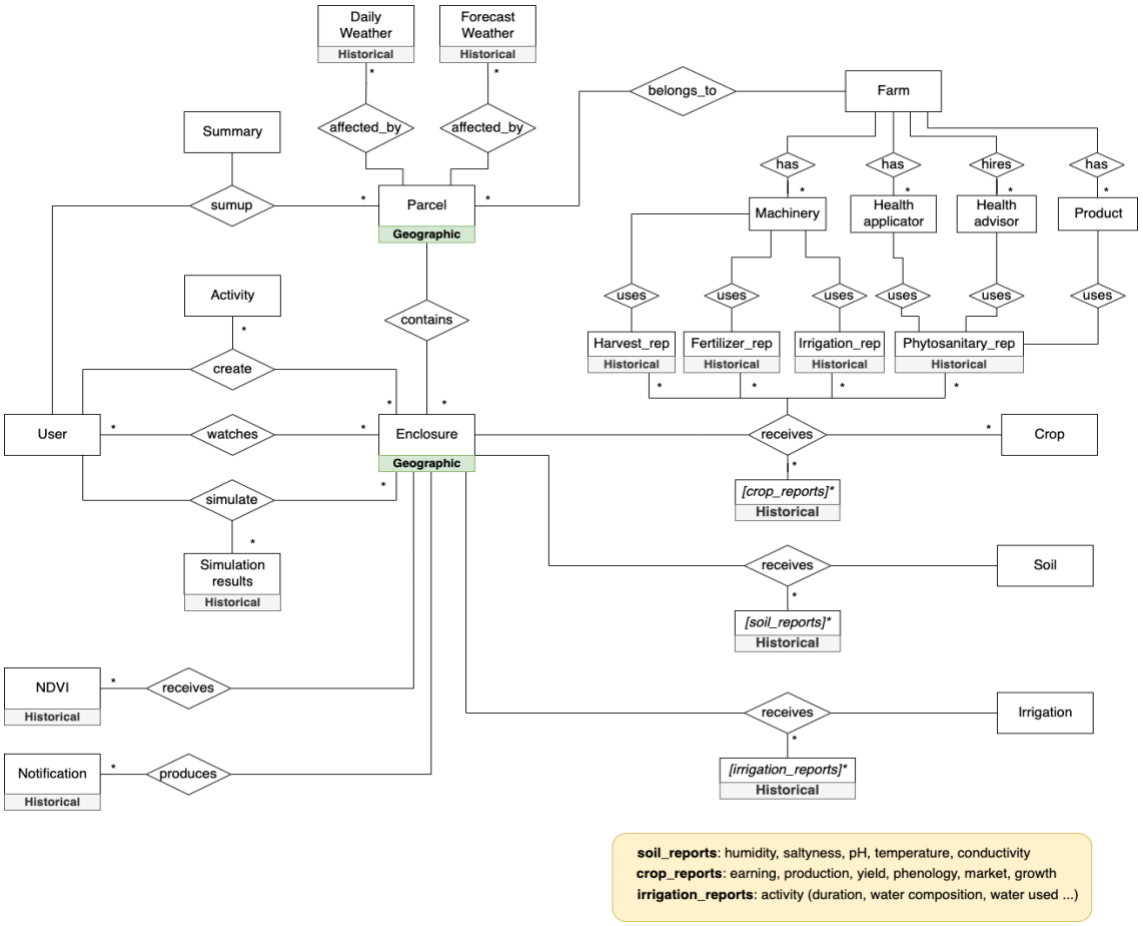


Ilustración 5: Diagrama ER simple.

A esto hay que añadir que dicho modelo de datos se encuentra en una primera fase, es flexible y se puede adaptar a nuevos cambios fácilmente gracias a la base de datos escogida para el *data mart*, MongoDB.

2.2.3 GEMELO DIGITAL

La arquitectura interna del servidor debe ser lo más flexible posible, debido a que las tecnologías que se van a manejar en un futuro pueden variar. Además, necesitamos un código limpio y mantenible que dure un largo periodo. Para ello, nos fijamos en los principios de la *clean architecture*[11], en concreto la variante de arquitectura hexagonal.

En el anexo III se explica más claramente qué es y por qué se ha usado pero, básicamente, permite mantener el dominio de la aplicación en el centro y define interfaces alrededor de él que dicen cómo el "mundo exterior" puede interactuar con él. En la Ilustración 6 se

[11] Es una filosofía de diseño software que permite separar los elementos de un diseño en anillos. https://blog.cleancoder.com/uncle-bob/2012/08/13 /the-clean-architecture.html

ve la arquitectura hexagonal aplicada a nuestro servidor. Esta arquitectura se explica mejor en la sección 4 del Anexo III.

MongoDB es la base de datos flexible y escalable por excelencia. Esta tecnología ha sido escogida precisamente porque el modelo de datos a usar no es robusto y sufrirá un buen número de cambios a lo largo de su vida. Incluso, se podría llegar a cambiar de tecnología de base de datos en un futuro. De esta forma, como *data mart*, usar MongoDB es una buena opción por el momento, aunque puede ser preferible una base de datos SQL una vez se tenga bien claro el modelo de datos a usar. Además del *data mart*, tenemos otros componentes dentro del componente de "Digital Twin" que se ve en la Ilustración 3: la caché de Redis, el servidor de gin-gonic y procesador ETL de Prefect.

El ETL de Prefect pretende integrar de forma periódica los datos en crudo del *data lake*, limpiándolos y realizando transformaciones para adaptarlo al modelo de datos del gemelo digital concreto, almacenándose en el *data mart*.

El caché de Redis no juega un papel tan importante en las primeras etapas, debido a la menor afluencia de peticiones y el caché HTTP que se integre. Sin embargo, cuando en un futuro se quiera abrir el sistema del gemelo digital a través de una API a otros sistemas o a través de un CLI, puede ser gran utilidad.

El servidor de gin-gonic es la entrada al sistema a través de la capa de presentación. Tiene tres tareas simples: hacer de *proxy* para el cliente, comunicarse con los servicios de la capa de lógica del gemelo y comunicarse con el bus de datos, que en este caso sería RabbitMQ. Principalmente se comunicará con el bus de datos cuando necesite información pedida por el cliente y que no se encuentre en el almacenamiento local.

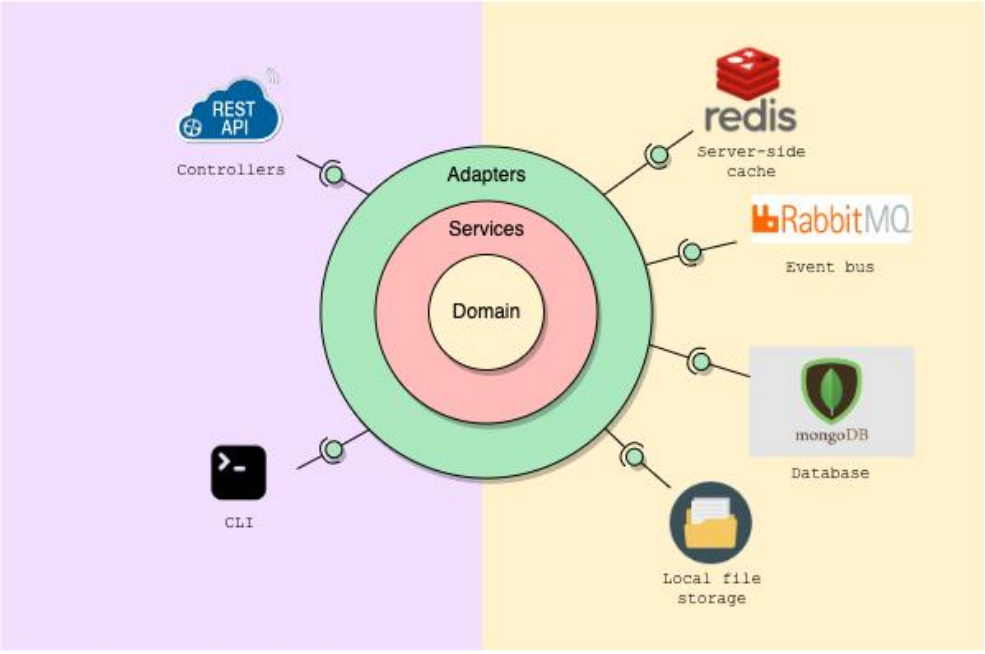


Ilustración 6: Arquitectura hexagonal aplicada al servidor principal de gin-gonic. Se pueden ver los adaptadores primarios a la izquierda (driving adapters) y los adaptadores secundarios a la derecha (driver adapters). Fase 1.

En este caso, el servidor también expone algunas imágenes usadas en la aplicación de forma estática. Dichas imágenes están, de momento, en el sistema de archivos de la máquina local. No obstante, si queremos cambiar el tipo de almacenamiento a un *bucket* de AWS, simplemente cambiaríamos el adaptador, sin tocar nada de lo que hay dentro de la capa de servicios y de dominio que se ven en la Ilustración 4, haciéndolo mantenible.

De nuevo, no se tiene en cuenta para esta fase del proyecto ningún tipo de análisis de los datos recogidos, dado que se trata de otra fase futura del proyecto.

2.2.4 GESTOR DE DATOS GLOBAL

Una de las interfaces que se implementan en dicha arquitectura hexagonal es la de la comunicación con el bus de datos que, si vemos la Ilustración 3, se corresponde con el bróker de mensajes RabbitMQ. Esta tecnología permite comunicar los diferentes gemelos digitales entre sí (en este caso sólo hay uno) y con el gestor de datos global, permitiendo un bajo acoplamiento tanto temporal como de mensajes[12].

Como hemos comentado en las secciones anteriores, el gemelo digital necesita alimentarse de datos, desafortunadamente, de distintas fuentes y en ocasiones en distintos formatos (JSON, XML, bytes …). Los datos pueden procesarse en fragmentos de tamaños diferentes y de una vez (*batch processing)* o en fragmentos más pequeños y de forma continua (*stream processing*). Principalmente, para esta primera fase, la información recolectada de los distintos sistemas es procesada en *batch*, debido a que son integraciones de datos bajo demanda o de forma periódica. De la misma forma, en fases posteriores se va a trabajar con datos procesados en *stream*, porque hay datos que llegan en tiempo real, o cada poco tiempo, de forma continuada. En ocasiones, los sistemas de los que extraemos los datos podrán ser sistemas legados, para los cuales es posible que necesitemos algún adaptador especial que, por ejemplo, Apache Camel puede contener.

Apache Camel nació como un *framework* de integración *open-source* muy versátil que permite diseñar rutas, que incluyen procesamiento ETL tradicional, usando los *enterprise integration patterns (EIP)*[13]*,* los cuales hemos mencionado en la sección 2 del artículo. Estos patrones, junto con las rutas de Camel nos permiten simplificar la integración de datos y convertirlo en un conjunto de *pipelines* que son sencillas de leer y de escribir y, por tanto, mantenibles en el tiempo. Camel cuenta con una gran cantidad de conectores para realizar las integraciones, lo que lo convierte en la tecnología perfecta para hacer procesamientos en *batch,* enviarlos al bus y/o guardarlos en el *data lake*.

[12] Más interdependencia en los módulos significa más acoplamiento. Menos interdependencia es menos acoplamiento. https://es.wikipedia.org/wiki/Acoplamiento_(informática)

[13] https://www.enterprise integrationpatterns.com

Servicios web bajo demanda

Existen algunos servicios que requieren una conexión directa entre el gemelo digital y dichos servicios, para reducir la complejidad de la misma. Por ejemplo, si el gemelo digital cuenta con un visualizador de mapas (leaflet p. ej.) o necesita acceder a un servicio web de mapas (mapas de NDVI p. ej.), es más sencillo que el gestor global de datos contenga las direcciones de los servicios y el gemelo digital pida esta información cuando la necesite. De esta forma, desacoplamos de nuevo el gemelo de la capa de gestión de datos.

Integración de datos de forma periódica

Desde Apache Camel, con la ayuda del componente "*Cron*"[14], programar rutas de integración de forma periódica, a la hora que se proponga. Obviamente, esto depende de la frecuencia de actualización de los datos de la fuente externa de la que se vaya a extraer. Una vez los datos son extraídos, se procesan mínimamente y se almacenan en el *data lake* en crudo.

Integración de datos bajo demanda

Esta se torna un poco más complicada, debido a que el flujo de información implica más componentes del sistema. Para explicarlo mejor, vamos a poner un ejemplo. Cuando un cliente quiere obtener la meteorología diaria de una parcela tiene que seguir el siguiente flujo:

1) El cliente web pide la información al *backend;*
2) El servidor de gin-gonic procesa la petición y pregunta directamente al componente "Global Data Management" (ya que no se almacena esa información localmente, sino el histórico con la información precisa) a través de una llamada RPC de RabbitMQ;
3) La ruta correspondiente de Apache Camel recoge la petición y mira en el *data lake;*
4) Si lo tiene, lo devuelve a RabbitMQ y de vuelta al controlador del servidor principal, que lo enviará al cliente;
5) Si no lo tiene, se va a buscar al servicio web de datos abiertos que nos lo proporcione;
6) Cuando se tiene, de forma paralela se guarda en el *data lake* con sus metadatos correspondientes y se envía;

En estos casos, es importante decir que, aunque la petición al gestor de datos global se desacopla y se desarrolla en paralelo, el servidor principal se queda bloqueado en la petición, obviamente, cancelando el contexto cuando pasa un *timeout*. Esto se realiza así por simplicidad (en futuras versiones se introducirá asincronía en este sentido). Si alguna vez se produce un *timeout,* es muy probable que la información extraída de la fuente externa se encuentre ya en el *data lake*. Por lo tanto, la nueva petición será rápida. Además, el cliente web contará con un caché HTTP en la mayor parte de las peticiones, debido a que no tienen que estar actualizándose constantemente. Y si es necesario, también se cuenta con caché a nivel de API con Redis.

[14] https://camel.apache.org /components/3.18.x/cron-component.html

Otros casos de integraciones de este estilo, guardarán los datos recogidos del bus de comunicación en el almacenamiento local, de forma que el flujo de información se reduce y la respuesta es más rápida.

Integración de datos en tiempo real

En este caso, estos datos llegan en tiempo real o con una frecuencia alta. Dichos datos serán transmitidos por algún tipo de middleware de mensajes como Kafka o RabbitMQ o incluso de bases de datos como Cassandra u otros. Estos mensajes son procesados por Apache Flink en *streamming,* siendo transferidos al *data lake* y/o analizados al vuelo de alguna forma que aún no está definida y/o enviados a alguna cola del bus de mensajes (RabbitMQ) para ser consumidos por el servidor principal, por ejemplo, que transmitirá la información al cliente a través de *streams* (como *Server Sent Events*[15]. Un ejemplo pueden ser el envío de información continua a través de los dispositivos de IoT.

Notificaciones y alertas en tiempo real

Aprovechando tanto las integraciones en tiempo real, el plan es tener algún tipo de motor lógico que inspeccione la información que llega al bus, de forma que se pueda analizar con respecto a otra información ya existente, o no. Con esto, se puede realizar un primer análisis a nivel del bus de datos, para inferir si puede existir algún tipo de irregularidad en los datos en tiempo real. De esta forma, poder reaccionar rápido ante un suceso potencialmente peligroso como, por ejemplo, sensores que den información por debajo o por encima de los umbrales permitidos, o un tractor autónomo que se sale de la ruta programada o se avería. Esto formaría parte de futura fases del gemelo.

En cuanto al *data lake*, para esta primera fase no se ha usado ninguna solución Cloud como Microsoft Azure, Hadoop o Apache Hive para el acceso más sencillo. En un futuro se tendrá que implementar por la gran cantidad de datos no estructurados o semi-estructurados que se querrán extraer. Por el momento, se ha levantado un clúster gratuito en el Cloud de mongoDB, que será más que suficiente.

En la sección 2.2.3, hemos hablado de cómo se ha implementado una arquitectura limpia como la arquitectura hexagonal que ayuda a mantener la limpieza y la mantenibilidad del código. Apache Camel, por su propia naturaleza, no necesita del uso de ninguna arquitectura especial. El flujo de datos funciona de la siguiente forma:

Conector → adaptador → procesamiento (dominio) → adaptador → conector

Se realiza una arquitectura por capas de una forma sencilla. Si se quiere cambiar un conector porque se va a utilizar otra tecnología, simplemente es cambiar el conector y su adaptador, tal y como se haría en la arquitectura hexagonal.

[15] Tecnología de *server push* que permite al cliente recibir datos del servidor a través de una conexión HTTP indefinida. Un ejemplo para Go: https://github.com/r3labs/sse

2.2.5 *MONITORIZACIÓN Y LOGGING*

En cuanto a la monitorización, se pueden ver en la Ilustración 3, en la zona superior izquierda las tecnologías que se han utilizado: Grafana, Prometheus y Loki. Realmente, por lo menos en estas primeras fases, no hacen falta tecnologías de indexación ni búsqueda de textos intensivos como Kibana, Logstash y Elasticsearch. Lo "único" que queremos hacer es monitorizar la aplicación de forma visual y sencilla, recogiendo métricas de los servicios del sistema y logs que nos resulten interesantes, principalmente en los flujos de datos con la zona del gestor de datos global. Más información en la parte de *devOps* de la aplicación en la primera sección del Anexo III.

No se han dibujado flechas para marcar de qué servicios se recopilan datos porque ensuciaría el diagrama de la Ilustración 3. Un ejemplo sería almacenar los logs de una ruta de Camel para ver si hay algún log con fallos o almacenar métricas del servidor principal para ver si hay errores 500 o 408, por ejemplo, y cuántos.

2.3 Dimensión del trabajo realizado

Desde un principio, el dominio del problema se planteaba complejo, por lo que los planes que se hicieron fueron prácticamente inexistentes. La información sobre el problema a tratar era escasa y un poco difusa, por lo que realizar un diseño se tornaba muy complicado. Por lo que se decidió diseñar un prototipo demasiado básico, que se puede ver en el Anexo III, sección 4. Además, no se pensó en un enfoque más escalable ni en el Big Data. Era una arquitectura cliente – servidor bastante simple, con su base de datos y su caché.

Por esto, tras un tiempo, se decidió realizar un rediseño de la aplicación, a aproximadamente dos meses antes de la entrega del proyecto, teniendo que pensar en nuevas tecnologías como Apache Camel, Apache Flink, RabbitMQ o Prefect, además de un cambio de visión hacia un modelo escalable y compatible con la nube. De ahí salió el modelo que se explica en la sección 2. Esta vez se diseñó un prototipo más realista que el anterior, más complejo y escalable (Anexo III, sección 4).

La planificación que se realizó estos dos últimos meses se corresponde con el diagrama de Gantt que se observa en la Ilustración 7.

03/10/2022 24/11/2022

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	TOTAL HOURS
Digital Twin	28	18							46
Frontend		13	10			6			29
Global data management			11	20	22	22			75
Enterprise service bus					4	8			12
E2E testing and technical report completion							15	18	33
TOTAL HOURS	28	31	21	20	26	36	15	18	195

Ilustración 7: Diagrama de Gantt correspondiente a la planificación final.

Las horas que aparecen en el diagrama de Gantt de la Ilustración 7 son las correspondientes a la última planificación. La cantidad de horas totales invertidas en el TFG son:

- Primera fase del TFG: 113 horas
  - Primer diseño del prototipo: 22 horas
  - Código e investigación: 91 horas
- Última fase del TFG: 221 horas
  - Diseño del prototipo final: 26 horas
  - Código e investigación: 195 horas (Ilustración 7)

Esto hace un total de 334 horas de trabajo.

2.4 Problemas encontrados

Como se ha explicado en la sección 2.3, el problema principal ha sido en torno al dominio del problema que, como dicho trabajo aborda un proyecto que puede tener un futuro a nivel industrial, se torna especialmente difícil diseñar una arquitectura base sobre la cual apoyar las siguientes fases de la aplicación. En sí, el modelo de datos es difuso, y se vuelve más complicado sabiendo que los sistemas externos de los que tomamos la información pueden ser tan heterogéneos.

Conectando con esto, al estar perdidos al intentar realizar un buen análisis del problema, se han cometido errores de diseño, como hacer una arquitectura demasiado simple o no escalable, como se requiere. A esto se le suma un primer diseño de interfaz pobre, que no tenía ninguna salida.

Junto a todo ello, está el tema de manejar y aprender nuevas tecnologías, como un mejor conocimiento de RabbitMQ, dándole un uso real, Apache Camel con su sintaxis y funcionalidades específicas o Prefect, dado que nunca se había usado Python para

programar. Además, un reto en cuanto a estándares aprendidos como GeoJSON, EIP o realizar un buen diseño arquitectural con la *clean architecture*.

3 LECCIONES APRENDIDAS Y CONCLUSIONES

3.1 Conocimientos adquiridos

Detrás de todos los problemas encontrados, se ha encontrado una solución, mejor o peor para solventar el problema. Este TFG ha sido un viaje por el descubrimiento de cómo funciona el inicio de un proyecto de un tamaño mediano-grande que es real y necesita ser convincente en su funcionalidad, ser escalable y ser mantenible en el tiempo. Esto implica saber confeccionar un diseño complejo, realizando un análisis inicial del dominio, con la poca información que pueda haber, e intentar hacer el sistema lo más modular y simple posible.

Además, un mayor conocimiento conceptual y tecnológico a la hora de imaginar la arquitectura del sistema, teniendo una visión más alejada del monolito tradicional de "Cliente – Servidor" simples hacia el diseño de una aplicación más asíncrona, ligera y desacoplada posible, siguiendo patrones de diseño importantes como *Facade*, *Adapter* o *Factory*, los cuales se pueden ver, por ejemplo, en la arquitectura hexagonal implementada. Además de esto, un mayor conocimiento en la inyección de dependencias en el código, que permite una mayor testabilidad de éste.

Junto a esto, se ha aprendido más profundamente cómo se realizan las integraciones de datos y las buenas prácticas a la hora de hacer las rutas y flujos de datos que se mueven en el *backend*. Los *enterprise integration patterns* que incluye Apache Camel son un conjunto nuevo de patrones que se ha descubierto, y que son de mucha utilidad. Unido, a esto, una visión más amplia de las fuentes de las que se puede integrar (servicios web, bases de datos, ficheros, otros sistemas), teniendo en cuenta que no toda información que existe en Internet se sostiene con tecnología actual, sino que hay mucho sistema legado.

Además, aprender a crear un diseño de interfaz más maduro, estandarizado a través de la reutilización de componentes y estilos, y con la información que se muestra bien escogida y repartida por las distintas rutas de la aplicación web. Además, para evitar una lluvia de peticiones al servidor, se ha investigado formas de cachear la información como, por ejemplo, a nivel de buscador (HTTP caché) y a nivel de servidor, con Redis.

Por último, no debemos olvidar una de los conocimientos aprendidos más importantes, como lo es la problemática de la que tratamos: la agricultura y sus procesos, para entender mejor el problema al que nos enfrentamos.

Se pueden ver todo esto más ampliamente en el Anexo III.

3.2 Trabajo futuro

Una vez realizado el prototipo para este TFG, tenemos que terminar lo que concierne a la fase 1 del proyecto, adquiriendo datos de todas las fuentes de información posibles e integrarlos, realizar *brainstorming* para pensar mejor la infraestructura y las relaciones de las variables existentes para general un modelo de datos más consistente.

Además, en cuanto a tecnologías, es posible que sea útil echar un vistazo a la arquitectura de microservicios, que puede tener sentido en cada instancia del gemelo digital si existen varios servicios que interactúan de forma frecuente y con almacenamiento para cada microservicio o conjunto de ellos. A esto se le añade la necesitar de tener en cuenta la migración de parte del sistema o en su totalidad a la nube, aunque sólo si se avanza mucho en el proyecto. Sin embargo, el sistema creado ya cuenta con capacidad de escalar fácilmente a través de Kubernetes, ya que sus componentes son compatibles.

Se insiste también en la necesidad de que otros sistemas puedan interactuar con el nuestro ofreciendo una interfaz al mundo (con autorización). Por ejemplo, se puede usar Apache Camel para crear una RestAPI para quien quiera extraer información o insertar información en nuestro sistema, añadiendo una pequeña interfaz a esta conexión privada.

En cuanto a la mejora de la parte de DevOps, de momento es algo sencillo, que pasa por la monitorización de logs y métricas, además del despliegue con Docker. En un futuro, se tendrán que hacer control de versiones en los gemelos digitales u orquestación y automatización de tareas y administración de configuraciones a través de herramientas como Ansible o Terraform. Además, lo que hemos dicho anteriormente en esta sección, investigar su despliegue completo para Kubernetes.

Por último, podemos mirar otra forma interesante de realizar la arquitectura de los gemelos digitales, como unir los gemelos digitales con la *blockchain* (Averay, 2020), para ganar un plus en seguridad, por ejemplo, y realizar ciertas transacciones a través de *smart contracts*, por ejemplo.

3.3 Conclusiones

En esta memoria se ha descrito cómo la arquitectura tradicional está siendo reemplazada a pasos agigantados por las nuevas tecnologías y la automatización de tareas, consiguiendo más eficiencia y siempre al servicio de las personas. Con este trabajo, se pretende buscar esta solución a través del software, sirviéndose de los datos ya existentes junto a datos en tiempo real como dispositivos IoT, de forma que se consiga modelar algo tan complejo como el comportamiento de un recinto agrícola. En este caso, modelizar recintos de frutos secos de cáscara, debido a que son especiales en comparación con el resto de cultivos (Anexo II).

El gemelo digital pretende suplir esa necesidad, alimentándose de grandes cantidades de información a través de la arquitectura Big Data, analizando dicha información en sus distintas formas, para obtener modelos predictivos, recomendaciones o informes sobre la actividad del recinto que quiere modelizar. Esto se hace con la necesaria ayuda del *machine learning,* a través de algoritmos de aprendizaje y predicción que permitan realizar simulaciones realistas sobre el recinto.

Este trabajo ha tratado de tener esto en cuenta, proporcionando un primer prototipo del proyecto, con una arquitectura compleja, pero eficiente, escalable y mantenible en el tiempo (Ilustración 2 y Ilustración 3). Como esto es sólo un prototipo, se debe tener en cuenta que sufrirá cambios a lo largo de su vida como proyecto. Sin embargo, como se apoya en soluciones actuales y en el uso de arquitectura limpia (sección 2.2.3), el mantenimiento pasa a ser mucho más sencillo.

Hemos conseguido conectar con éxito una instancia de gemelo digital simple con el gestor de datos global, para integrar distinta información sobre las parcelas que se manejan de sistemas externos (integración periódica) en el *data lake*, además de pedir información al gestor de datos global bajo demanda (mirar en el *data lake* y/o buscar la información en servicios web). Todo esto, a través del "Enterprise service bus", desacoplando los componentes del sistema. De la misma forma, hemos realizado trabajos de ETL más potentes para extraer información en crudo del *data lake* y enviarla, transformada, al *data mart* del gemelo digital. Además, se han monitorizado dichas acciones a través de métricas y logs, y visualizados en un panel de forma cómoda. Junto a esto, se ha realizado un panel de administración de parcelas (aplicación web) para visualizar esta información.

Con esta arquitectura, se consigue abstraer de forma exitosa un tipo de problema como es el de la modelización de activos físicos, difíciles de predecir en su comportamiento, y generalizarlo en instancias que denominamos gemelos digitales. Así, se consigue proporcionar al usuario un sistema de monitorización, de toma de decisiones y de simulación para predecir de forma efectiva el comportamiento de los cultivos de un recinto.

BIBLIOGRAFÍA

sadsma. (s.f.). Obtenido de sadsma: http://www.sadsma.cdmx.gob.mx:9000/datos/glosario-definicion/Agricultura

El cultivo de frutos secos, una apuesta rentable. (3 de Agosto de 2016). *interempresas*. Obtenido de https://www.interempresas.net/Horticola/Articulos/185586-El-cultivo-de-frutos-secos-una-apuesta-rentable.html

¿Cuál es la diferencia entre una simulación y un gemelo digital? (Noviembre de 2020). *fegemu solutions*. Obtenido de fegaut: https://www.fegaut.com/es/blog/cual-es-la- diferencia-entre-una-simulacion-y-un-gemelo-digital/0-432/

Población mundial actual. (s.f.). Obtenido de worldometer: https://www.worldometers.info/es /poblacion-mundial

Kaloxylos, A., Eigenmann, R., Teye, F., Politopoulou, Z., Wolfert, S., Shrank, C., . . . Alonistio, N. (2012). Farm management systems and the Future Internet era. *Computers and Electronics in Agriculture*, 130-144.

Big data: definición, tipos, características y beneficios. (s.f.). *Postgrado UCSP*.

Rathore, M. M., Shah, S. A., & Shukla, D. (2021). The Role of AI, Machine Learning, and Big Data in Digital Twinning: A Systematic Literature Review, Challenges, and Opportunities. *IEEE Access*.

Rasheed, A., Sauerwein, D., & Gounteni, a. S. (17 de Junio de 2022). Digital Twins on AWS: Predicting “behavior” with L3 Predictive Digital Twins. *AWS Official Blog*.

¿Qué es la informática sin servidor? (31 de Octubre de 2017). *RedHat*.

¿Qué son y para qué sirven los microservicios? (9 de Marzo de 2018). *RedHat*.

Lawton, G. (s.f.). *techtarget*. Obtenido de techtarget: https://www.techtarget.com/searchdatamanagement/definition/stream-processing

AWS. (s.f.). Obtenido de AWS: https://aws.amazon.com/what-is/batch-processing/

Averay, M. (23 de Noviembre de 2020). Blockchain & The Digital Twin. *Medium*.

NDVI: Índice De Vegetación De Diferencia Normalizada. (s.f.). Obtenido de eos: https://eos.com/es/make-an-analysis/ndvi/

ANEXO I. GEMELOS DIGITALES EN AGRICULTURA

Los Gemelos Digitales (GD) son réplicas digitales de sistemas físicos reales (vivos o no) en los que se da cabida a la modelización del sistema físico al que se refieren entrelazando soluciones de análisis de sistemas complejos, soporte de decisiones e integración de tecnología de diferente naturaleza. Los GD han ganado prominencia, en parte, gracias a la adopción de tecnologías de Internet de las cosas que permiten la "alimentación" de datos de estas representaciones de sistemas físicos, a altas resoluciones espaciales, casi en tiempo real, a través de dispositivos en miniatura y sensores remotos, proveyendo de flujos de datos cada vez mayores. Los GD están siendo adoptados cada vez por más actores de la cadena productiva, incluidos los sectores de fabricación, automotriz, energía, y, en menor medida, agricultura.

Pylianidis[16] presenta una revisión de la literatura sobre gemelos digitales en la agricultura, que abarca los años 2017-2020. En este trabajo se identifican 28 casos de uso, y se compara con la adopción de este paradigma en otras disciplinas. A partir de aquí, se identifican dos características distintivas de GD en la agricultura:

- Los GD agrícolas involucran directa o indirectamente sistemas vivos y productos perecederos. Si bien los GD son ideales para proporcionar información sobre sistemas tan complejos e incorporar procesos no deterministas, su integración con el gemelo físico puede ser difícil. Esto se amplifica aún más debido a la idiosincrasia de los gemelos físicos vivos.
- Los GD agrícolas tienen una dimensión espacio-temporal de su funcionamiento mayor que la de los GD en otras áreas industriales. El GD en otras disciplinas oscila entre el tamaño de un avión y el de una fábrica. Los GD agrícolas van desde plantas y animales individuales, hasta parcelas de tierra, granjas o regiones gemelas. Adicionalmente, los GD agrícolas difieren de otros GD en que sus tasas de respuesta son más lentas. Los procesos agrícolas como el cultivo de plantas, tienden a evolucionar de forma relativamente lenta, por lo que, al menos inicialmente, no hay necesidad de interacciones de alta frecuencia entre gemelos físicos y digitales.

El segundo de estos elementos diferenciales de los GD agrícolas resulta especialmente atractivo al sector que comprende la naturaleza laboriosa de monitorear, predecir y controlar la salud de los árboles frutales, de las herbáceas de temporada, o de cualquier otro cultivo en el que exista un amplio distanciamiento entre el momento de la siembra y el de la cosecha, con todos los condicionantes que ello supone para el control y mejora de la calidad de la misma. Por ejemplo, un equipo de la Universidad de Queensland ha desarrollado un GD para una explotación de cultivos de crecimiento lento como mango

[16] C. Pylianidis, S. Osinga, I. N. Athanasiadis, "Introducing digital twins to agricultura", Computers and Electronics in Agriculture, Volume 184, 2021, 105942, ISSN 0168-1699, https://doi.org/10.1016/j.compag.2020.105942.

y macadamia[17]. Sobre el mismo, se puede permitir a los usuarios probar rápidamente nuevas ideas y obtener información sobre cómo optimizar mejor los sistemas de producción, sin tener que esperar a que las plantas lleguen a la edad de producción. En este punto, la presente propuesta de proyecto hace suyo este planteamiento de abordar la "gemelización" de explotaciones como el pistacho (empieza a producir a partir del tercer o cuarto año, aunque no llega a máxima producción hasta el octavo año), la almendra (empieza a producir a partir del segundo año, aunque no llega a máxima producción hasta pasado el quinto año), o la nuez (empieza a producir a partir del quinto año, aunque no llega a máxima producción hasta el trigésimo año).

Por otro lado, cuando se trata de procesos agrícolas, el uso de GD como medio central para la gestión agrícola puede permitir el desacoplamiento de los flujos físicos de su planificación y control. Como resultado, los agricultores pueden administrar las operaciones de forma remota basándose en información digital (casi) en tiempo real en lugar de tener que depender de la observación directa y las tareas manuales sobre el terreno. Esto les permite actuar inmediatamente en caso de desviaciones (esperadas) y simular los efectos de las intervenciones basándose en datos de la vida real. Por ejemplo, un GD de un huerto podría alertar al huerto de riego excesivo sin que el agricultor tenga que examinar el huerto "real".

[17] https://www.goodfruitandvegetables.com.au/story/7413783/digital-modelling-to-influence-orchard-design/

ANEXO II: EL FRUTO SECO DE CÁSCARA

Este anexo hace un breve análisis del fruto seco de cáscara como cultivo relevante para llevar a cabo un trabajo de tecnificación que aspire a resultar rentable, así como qué es lo que hace que el mismo necesite de un tratamiento separado de otros cultivos desde el punto de vista de la digitalización de sus explotaciones.

La producción española de fruto seco de cáscara

Europa occidental constituye el principal mercado para los frutos secos. El nivel de demanda ha sido creciente en los últimos años, sin embargo, la producción europea no ha seguido el ritmo de Estados Unidos, el cual se ha convertido en el principal proveedor de frutos secos del mercado europeo. La fruticultura europea se ha estado desarrollando básicamente en zona húmeda o en regadío. No obstante, algunas especies de frutos secos son susceptibles de cultivo en condiciones de mayor aridez, lo que puede suponer una alternativa de futuro en amplias áreas de países mediterráneos. Por ello, y la creciente demanda en Europa, las posibilidades de desarrollo del cultivo para producción de fruto seco en España son elevadas.

El consumo actual español de frutos secos y la capacidad transformadora industrial es muy superior a la producción. España fue en tiempos una gran potencia productora de almendra, pero el liderazgo mundial lo tiene actualmente California, donde se ubica la entidad de referencia mundial *Almond Board*. Estados Unidos concentra el 80% de la producción mundial de almendra, que se calculó en 1.684.395 toneladas (t) en 2020.

Almendrave (Spanish Almond Board) cifraba la superficie española dedicada al almendro en 718.000 hectáreas en 2019, frente a las 527.000 en 2014. También se apuntaba que el regadío, que puede producir hasta siete veces más que el secano, va camino de representar la mitad de la superficie; y se calculó que el 25% está en producción ecológica y que la quinta parte de lo plantado aún no ha entrado en producción. Sin embargo, el avellano ha retrocedido de 33.000 a 13.000 has. La producción este año puede rondar las 110.000 t de almendra grano, pero en los próximos años puede alcanzar las 180.000 toneladas. En España, la previsión de cosecha de almendra, por ejemplo, se estima en 353.705 t, para la campaña 2020-2021, un 4% superior a la campaña anterior. Por su parte, para la avellana, se prevé que alcance las 12.324 t, suponiendo un 1.6% más que la campaña anterior[18].

Si nos referimos a la demanda nacional, los frutos secos tienen un importante uso en la industria agroalimentaria. Centrándonos en el hogar, según el Informe de Consumo Alimentario en España 2020 del Ministerio de Agricultura, Pesca y Alimentación, la evolución ha sido favorable en el último año con crecimientos cercanos al 20%. Los frutos secos suponen 0,55% del volumen de compra, pero el 1,66% del gasto, con una media de 28,45€/per cápita, cercano por ejemplo a un producto tan común en los hogares como

[18] Ministerio de Agricultura, Pesca y Alimentación, Nota de Prensa, 22/09/2020.

el aceite de oliva. En hogares se consumen más nueces, seguidas de cacahuetes, almendras y pistachos. Los frutos secos tienen un valor bruto de 1.315 millones de euros y 176.265 t. Las perspectivas son de aumento de la demanda, ayudado por unas propiedades saludables que el consumidor valora cada día más.

Los cuatro principales frutos secos cultivados en España son: almendro, avellano, nogal y pistachero. Cada especie presenta una situación y unas perspectivas de desarrollo diferentes. Tradicionalmente, el almendro y el pistachero se han considerado cultivos de secano mientras el avellano y el nogal lo han sido de regadío. Esta situación está cambiando, en la mayoría de las condiciones agroclimáticas españolas, y actualmente los cuatro frutales requieren para una producción competitiva dosis suficientes de agua lo que, en determinadas ubicaciones ha supuesto un cambio del modelo agronómico, incluso llegando a desaparecer en determinadas zonas.

Sin embargo, en la actualidad se está impulsando el renacer de estos cultivos, en especial el almendro y el pistacho. Existen cuatro causas principales que están favoreciendo este impulso: (1) la falta de rentabilidad de los campos, especialmente de cereal, que hace que las explotaciones se diversifiquen hacia frutos secos; (2) altas cotizaciones de las producciones de frutos secos en los últimos años; (3) alto grado de mecanización que se ha implantado, adoptando nuevas técnicas y maquinarias, lo que supone un abaratamiento de costes, y; (4) los resultados de investigaciones de nuevas variedades más tardías, resilientes y productivas que hacen que prosperen árboles en áreas anteriormente inapropiadas.

Al igual que en el resto de las explotaciones agrarias de España, los sistemas de producción de frutos secos están actualmente experimentando una revolución tecnológica debido a la gran demanda del mercado y a la moderación de sus precios. Existe una tendencia generalizada hacia la intensificación del cultivo y la reducción de sus costes de producción, siendo muy importante en estos cuatro cultivos la recolección y las operaciones de secado y postcosecha. Los precios, no obstante, vienen determinados por las producciones anuales en los principales países productores: EEUU (almendra, pistacho y nuez), Irán (pistacho), Turquía (avellano) y China (nuez) y las cotizaciones euro/dólar.

Actualmente, los frutos secos se muestran como una importante alternativa a otros frutales y tienen posibilidades en:

- Producción ecológica,
- Lucha contra la despoblación rural,
- Beneficios contra el cambio climático,
- El mantenimiento del paisaje.

La producción española de frutos secos destaca por su calidad frente a la de otros países productores.

El fruto seco de cáscara frente a otros cultivos

Los frutos secos de cáscara cuentan con varios elementos diferenciales de otros cultivos que llevan a su consideración por separado desde el punto de vista de modelización de sus explotaciones. De una parte, se obtienen de árboles frutales que requieren una etapa inicial de cuidado no productivo. Esto marca una diferencia sustancial frente a cultivos que únicamente tienen una o varias cosechas, y que empiezan a resultar productivos el primer o el segundo año después de su plantación. En estos cultivos estamos, entre otros, considerando a los cereales (cultivos de plantación y una única cosecha), leguminosas (como por ejemplo la alfalfa que puede ofrecer cosecha durante 5 años), y todo tipo de hortalizas.

Dentro de los árboles frutales, el fruto seco de cáscara tiene también elementos diferenciales para su consideración por separado.

- Los frutos secos, por su propia definición, son aquellos que carecen de jugo. Se trata de frutos con una cáscara muy dura que presentan un porcentaje de agua inferior al 50%. Esto hace que sea posible el cultivo industrial de los mismos sin sistemas de regadío (bajo la denominación de cultivo de secano). Esta modalidad de cultivo no resulta industrialmente relevante cuando se trata de otros árboles frutales (los generalmente conocidos como frutos de hueso o de pepita).
- La mencionada limitada presencia de agua hace que el tipo de problemas fitosanitarios sea diferente y las condiciones de riesgo de los mismos también. Por ejemplo, una gran presencia de humedad ambiental supone una situación de muy alto riesgo de plaga para la mayor parte de los frutos secos, especialmente en el caso de los frutos secos de cáscara.
- La necesidad de que las explotaciones de frutos que no son secos se base en regadío hace que el modelo de fertilización más habitual sea el de fertirigación. Por el contrario, en el caso de los frutos secos la fertilización se hace generalmente mediante maquinaria especializada, separada de los procesos de regadío. Así mismo, en muchos casos, los ritmos de fertilización son distintos para los frutos de hueso y pepita (anual y, en ocasiones, en varias aplicaciones), frente a los secos de cáscara (en muchas ocasiones una fertilización cada dos o más años).
- En los frutos secos de cáscara, el principal valor industrial está en la semilla que, además, está protegido por la cáscara. Debido a esto, la cosecha está completamente soportada por maquinaria que, en muchos casos, ya están sensorizadas. Frente a esto, la cosecha de otros árboles frutales, a día de hoy, se efectúa principalmente a mano ante la necesidad de proteger al fruto (posiblemente la oliva es de las pocas frutas en las que ya se incorpora la maquinaria sensorizada para su cosecha).

ANEXO III: UN POCO MÁS DEL PROTOTIPO

En este anexo se van a explicar de forma más detallada los aspectos más técnicos del prototipo diseñado.

Tecnologías más en profundidad

Como hemos contado en la sección 1.2 de la memoria, se han hecho uso de varias tecnologías, cada una con su propósito claro. Ahora, para cada una de las tecnologías que se han escogido, se explica el porqué de dicha elección sobre otras tecnologías:

- Golang (gin-gonic) permite realizar programas concurrentes de forma muy sencilla y poco pesada. Golang es un lenguaje rápido y es ampliamente utilizado y apoyado por Google. Cosas como las *goroutines* y los canales, hacen muy sencillos el uso del *multi-threading*. Para el servidor principal, necesitamos un componente altamente concurrente y ligero, para poder soportar buena cantidad de peticiones con un ahorro de recursos considerable en relación a NodeJS o Python.
- Apache Camel es de los pocos frameworks de integración que es bueno, permite el despliegue en varias tecnologías y es tan completo. En este caso, usamos Kotlin como lenguaje de programación, por su sencillez en comparación a Java y Quarkus como tecnología de Java nativa en Kubernetes que, siendo muy similar a Spring Boot, ofrece mayor rendimiento y velocidad[19]. Comparando con otros como MuleESB o Spring Integration, Apache Camel tiene las de ganar, con su amplia comunidad y su recorrido de más de 10 años[20].
- Prefect es un orquestador de *workflows* de Python para realizar *pipelines* ETL, en este caso, del *data lake* que tenemos con datos en crudo a los *data marts* de los distintos gemelos digitales. En este caso, la opción no se ha comparado exhaustivamente, simplemente funciona y la sintaxis es muy sencilla. Como es un orquestador, podemos usar las librerías de ETL o *machine learning* que queramos, como Pandas o Numpy.
- RabbitMQ es un bróker de mensajes que se caracteriza por su enrutado más complejo y manejable, en comparación con otros middlewares de este estilo como Kafka. Se ha escogido principalmente por su popularidad, además de su capacidad de escalar verticalmente y de forma sencilla. Kafka era una alternativa demasiado pesada para este caso de uso.
- Apache Flink es un framework de procesamiento distribuido para computaciones con estado, manejando *bounded* y unbounded *streams*. Se ha elegido en un principio porque, además de que lleva con nosotros más de 10 años, es más ligero y consume menos ancho de banda que otros como Apache Kafka o Apache Spark. En cuanto a esta tecnología, se podrá saber más en futuras versiones.

[19] https://refactorizando.com/spring-boot-vs-quarkus/

[20] https://www.techtarget.com/searchapparchitecture/tip/Apache-Spring-and-Mule-Explore-3-top-integration-frameworks

- Svelte es un framework de Javascript que, lejos de ser una librería de Javascript como ReactJS o AngularJS, es un compilador de Javascript. En un principio, se empezó con el uso de ReactJS por la experiencia que se tenía. Sin embargo, cuando se hizo el rediseño del sistema, se eligió Svelte debido a su simplicidad, su modularidad y su, más fácil sintaxis y legibilidad, además de su rendimiento superior, debido a que no usa un *VirtualDOM*, sino que actúa directamente sobre el DOM real[21].

En cuanto a la parte de *devOps,* contamos con Docker para lanzarlo, el cual, de momento, es más que suficiente para meterlo todo en contenedores de forma que se pueda desplegar en cualquier servidor de forma sencilla. Para esta primera versión, contamos con una máquina en la que desplegamos el Docker y, a través de un servicio de Nginx que hay en la máquina se puede exponer al mundo por el puerto que se le configure. También contamos con un certificado de Let's Encrypt para asegurar la comunicación HTTPS. En el Ilustración 8 podemos ver la configuración usada para Nginx (es temporal), de forma que tenemos una ruta para que *certbot* gestione la renovación del certificado Let's Encrypt.

```
server {
    listen 80;
    listen [::]:80;
    location /.well-known/acme-challenge/ {
        root /var/www/certbot;
    }
    location / {
        root /usr/share/nginx/html/lp;
        index  index.html;
        try_files $uri $uri/ /index.html;
        add_header Cache-Control 'no-store, no-cache, must-revalidate, proxy-revalidate, max-age=0';
    }
        ...
```

[21] https://pagepro.co/blog/react-vs-svelte/

```
}
```

Ilustración 8: Parte de la configuración de Nginx para la aplicación web

Listado completo de requisitos

Ahora, pasamos a detallar los requisitos funcionales del sistema creado, junto a un pequeño diccionario de palabras para aclarar mejor algunos conceptos descritos en los requisitos.

Diccionario de palabras:

1. Sistema: se define sistema como la parte de *backend* de la arquitectura creada que, si miramos en la Ilustración 2, englobaría todo, incluyendo el gestor de datos global y los gemelos digitales.
2. Fuentes de datos: datos abiertos (meteorología, fitosanitarios, fertilizantes), datos de ERPs o servicios web, de otros FMIS, de sistemas *operation technology* (OT) (sensores, maquinaria, comandos (5), actividades), bases de datos y otros (Big Data).
3. De forma desacoplada: cuando decimos que algo se realiza de forma desacoplada nos referimos a realizar la tarea sin que las dos partes que se intercambian información se conozcan de forma directa y/o de forma asíncrona. Para esto se ha diseñado el "Enterprise Service Bus" de la Ilustración 2.
4. Análisis de la información: predicciones, estadísticas, recomendaciones, *machine learning*, relaciones, étc.
5. Comandos: operaciones que se envían a los elementos IoT de forma que realicen un trabajo pedido (regar a una determinada hora, aplicar fertilizantes o fitosanitarios, manejar maquinaria de forma remota)
6. Simulaciones: a través de los modelos predictivos que se tengan, usar predictores y *machine learning* para extraer resultados interesantes, como cuándo se debería cultivar una planta, cuándo es el mejor momento para cosechar, para regar, étc.

Los requisitos funcionales que se detallan en la Tabla 1 están enfocados en la infraestructura que se quiere crear y cómo debería funcionar. Hay que decir que es posible que falten requisitos, los cuales se completen a medida que se le den más vueltas al proyecto. Además, dichos requisitos siguen un alto nivel, dado que se podrían dividir en subrequisitos más concretos.

RF	Descripción
RF1	El sistema (1) debe ser capaz de integrar información de distintas fuentes de datos (2), de forma que se puedan "alimentar" los distintos gemelos digitales
RF2	El sistema debe ser capaz de almacenar la información extraída de forma permanente para no depender de sistemas terceros en un futuro
RF3	El sistema debe ser capaz de permitir a otros sistemas externos la comunicación con nuestro sistema, ya sea para extraer o insertar información de forma controlada y autorizada
RF4	El sistema debe ser capaz de realizar una primera limpieza para evitar fallos en la información de los datos recibidos de las distintas fuentes de información
RF5	El sistema debe ser capaz de realizar un primer análisis (4) de los datos que se reciben de las distintas fuentes de datos para extraer información útil
RF6	Los recursos del sistema deben estar protegidos por un sistema de RBAC, de forma que haya restricciones en el acceso a dichos recursos
RF7	Los componentes del sistema deben estar monitorizados en todo momento, a través de recolección de métricas y logs
RF8	Un gemelo digital debe ser capaz de extraer información a través del sistema diseñado de forma desacoplada (3) para procesarla y almacenarla de forma local (ETL)
RF9	Un gemelo digital debe ser capaz de comunicarse con otros gemelos digitales de forma desacoplada
RF10	Un gemelo digital debe realizar una serie de análisis (4) de la información ya limpia de su almacenamiento local, de forma que se puedan extraer *insights*, estadísticas y relaciones interesantes de los datos
RF11	Un gemelo digital debe ser capaz de permitir la petición de datos bajo demanda a través del sistema, de forma desacoplada
RF12	Un gemelo digital debe ser capaz de enviar comandos (5) a los sistemas externos
RF13	Un gemelo digital debe ser capaz de realizar simulaciones (6) bajo demanda de los recintos que monitorice, extrayendo información útil de éstas
RF14	El sistema debe ser capaz de generar gemelos digitales bajo demanda
RF15	Un gemelo digital debe ser capaz de acceder a servicios web de forma sencilla y desacoplada
RF16	El sistema debe generar notificaciones relevantes en tiempo real de lo que sucede en un recinto monitorizado
RF17	El sistema debe tener un mínimo nivel de reconfiguración al cambiar las condiciones y entrar nuevos datos al sistema *(necesita pensarse mejor)*

Tabla 1: Listado de requisitos funcionales.

Guía de estilo

El *backend* diseñado, en el cual nos hemos centrado, guarda los datos necesarios para que la aplicación web que tenemos los muestre al usuario. De momento, esta aplicación ha sido diseñada como un panel de administración de las parcelas que se quieren monitorizar.

Está creada con Svelte, que nos permite hacer un desarrollo más simple, limpio y escalable. Esta tecnología se ha elegido, entre otras opciones, gracias al árbol de decisión de tecnologías de la Ilustración 9.

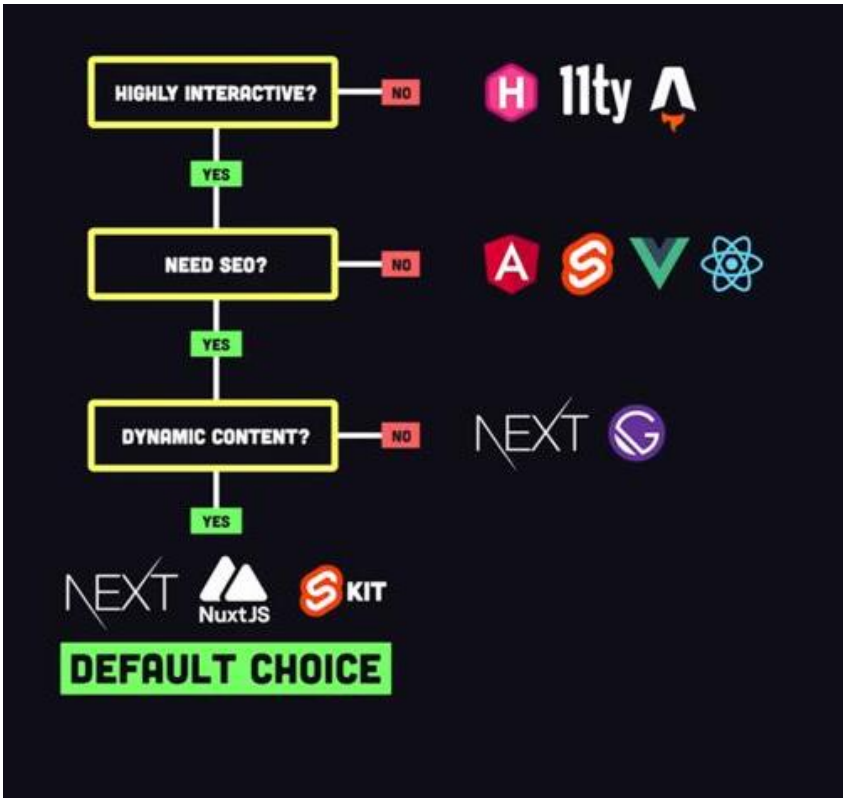


Ilustración 9: Árbol de elección de tecnologías de frontend. Fuente: Fireship.io

En cuanto al estilo usado en la aplicación web, los colores usados son los que se muestra en la Ilustración 10, siendo el verde el color principal y el naranja el color secundario.

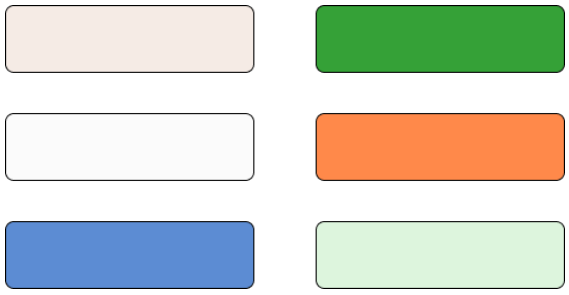


Ilustración 10: Colores usados para la aplicación

En un principio, la interfaz diseñada para el prototipo lucía como las figuras que se muestran a continuación.

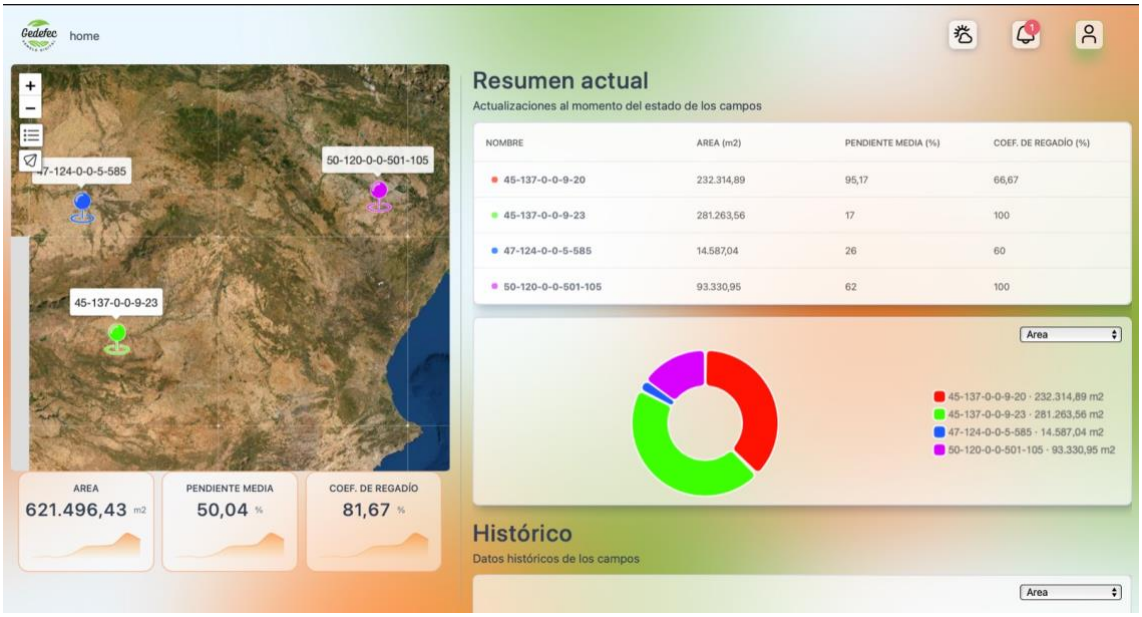


Ilustración 11: Overview. Donde se visualizan las parcelas y las características comunes

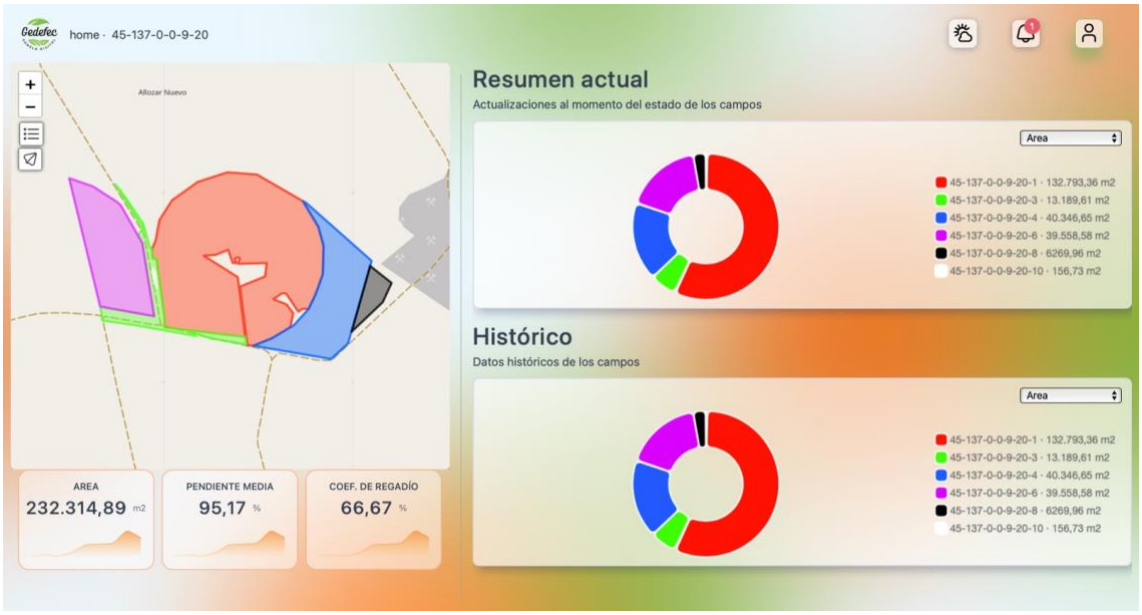


Ilustración 12: Parcela concreta

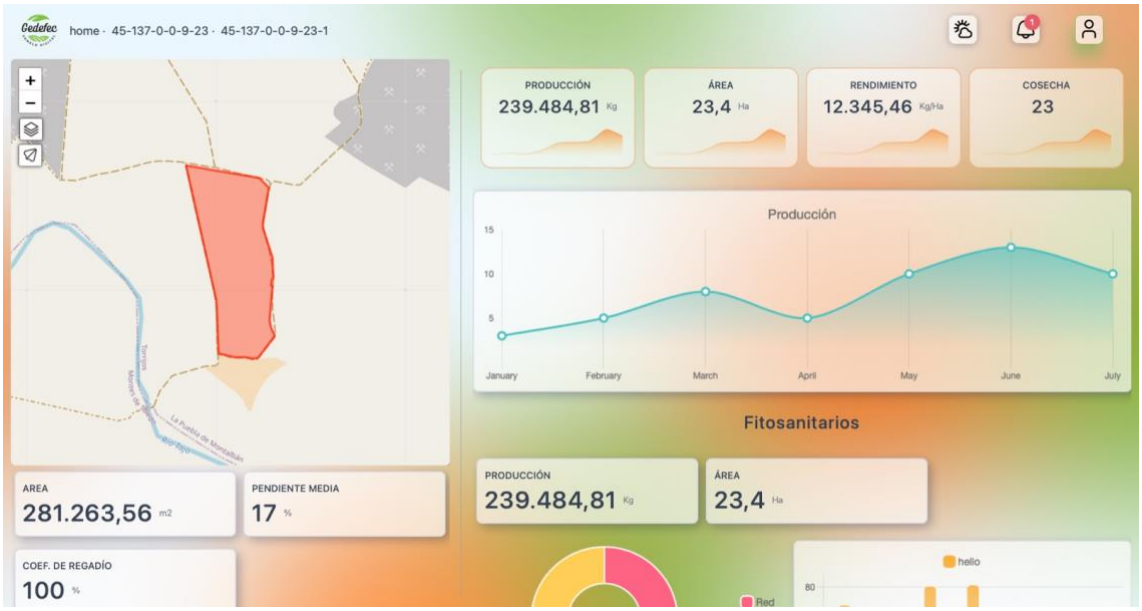


Ilustración 13: Recinto concreto

Estos diseños son ya parte del prototipo funcional que se tenía hasta antes de hacer el rediseño. Falla especialmente en la organización de las páginas, y en pecar de simplista en su diseño, intentando embeber todo en tres páginas principales. En rediseño nuevo se visualiza en las siguientes figuras.

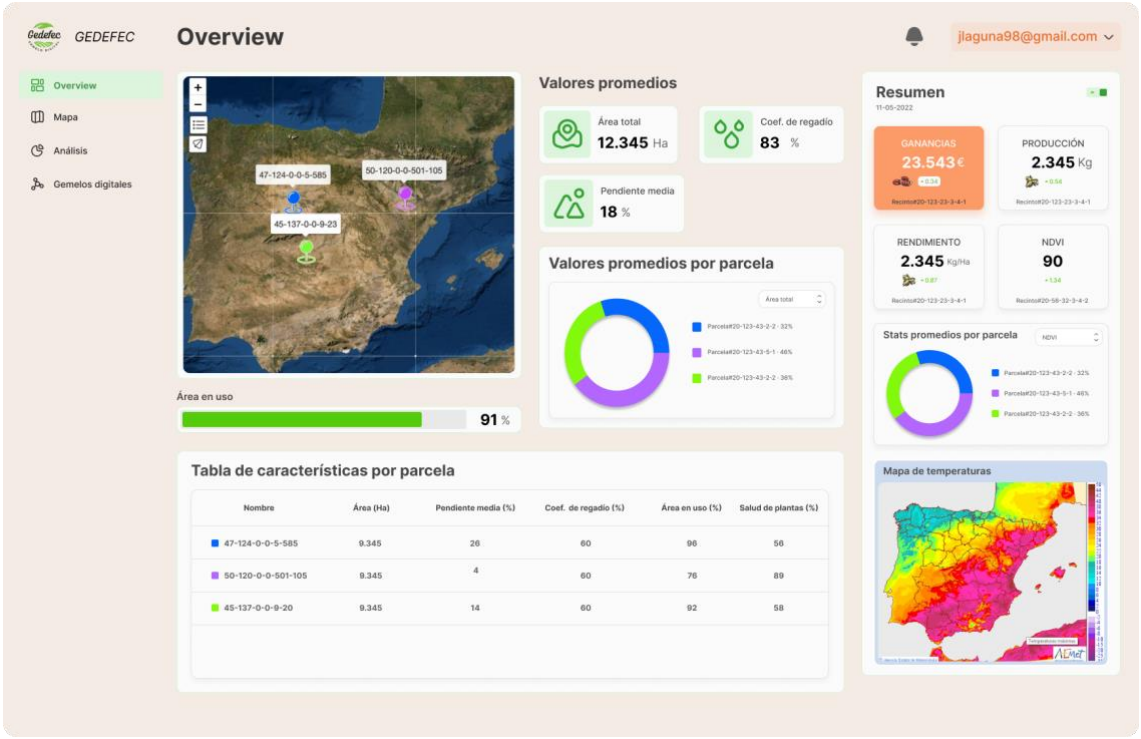


Ilustración 14: Overview. Donde se visualizan las parcelas, características comunes, resumen y el tiempo

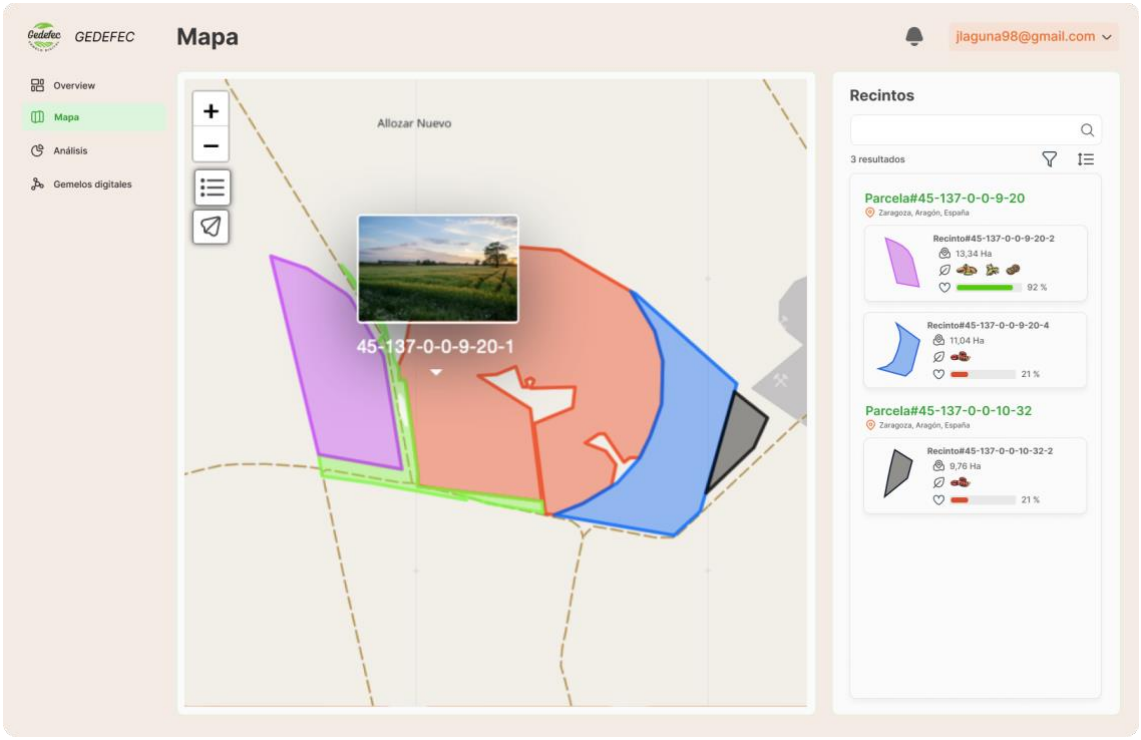


Ilustración 15: Mapa. Donde se visualizan todos los recintos en un mapa y pueden visualizarse en forma de lista

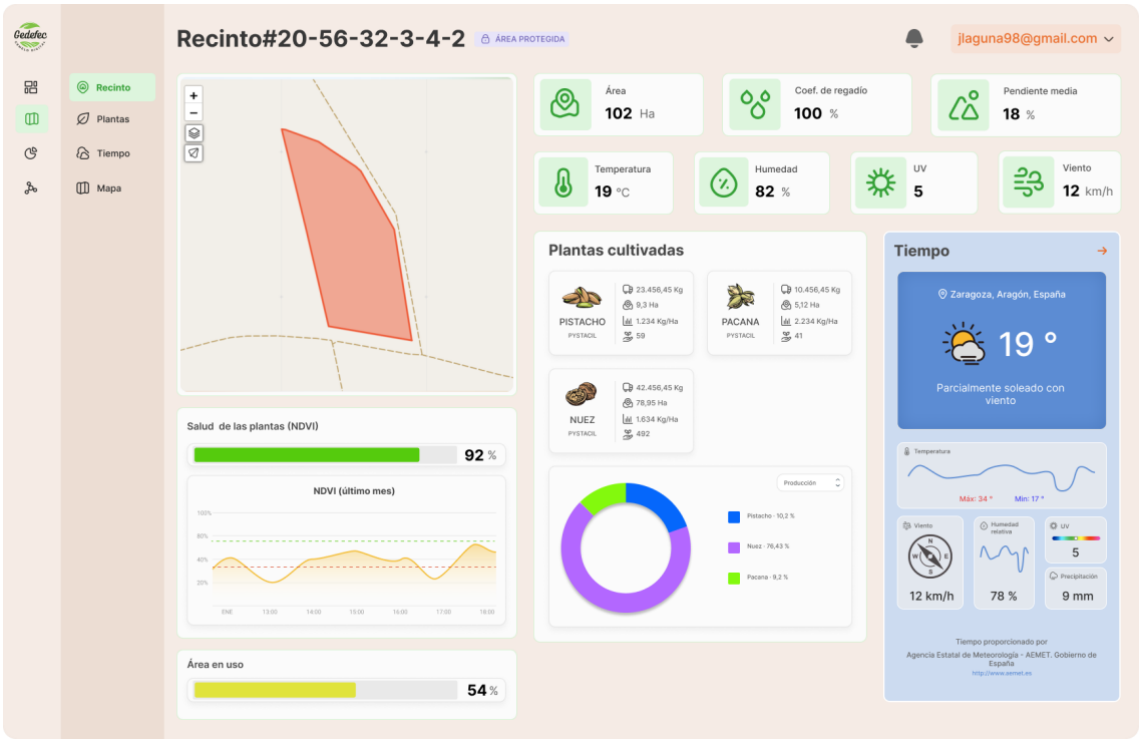


Ilustración 16: Overview de un recinto. Aquí se ven las características, los datos de los sensores, el tiempo de la parcela, NDVI, las plantas y más

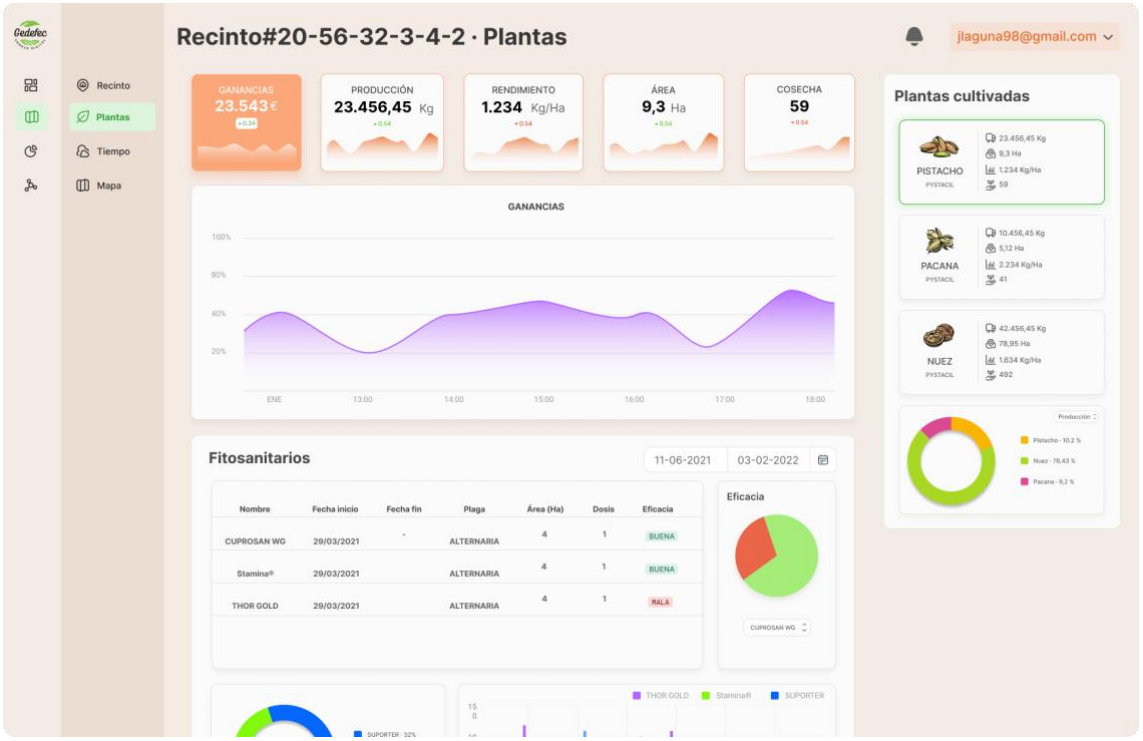


Ilustración 17: Plantas de un recinto. Información sobre ganancias, rendimiento, fertilizantes, fitosanitarios y otros

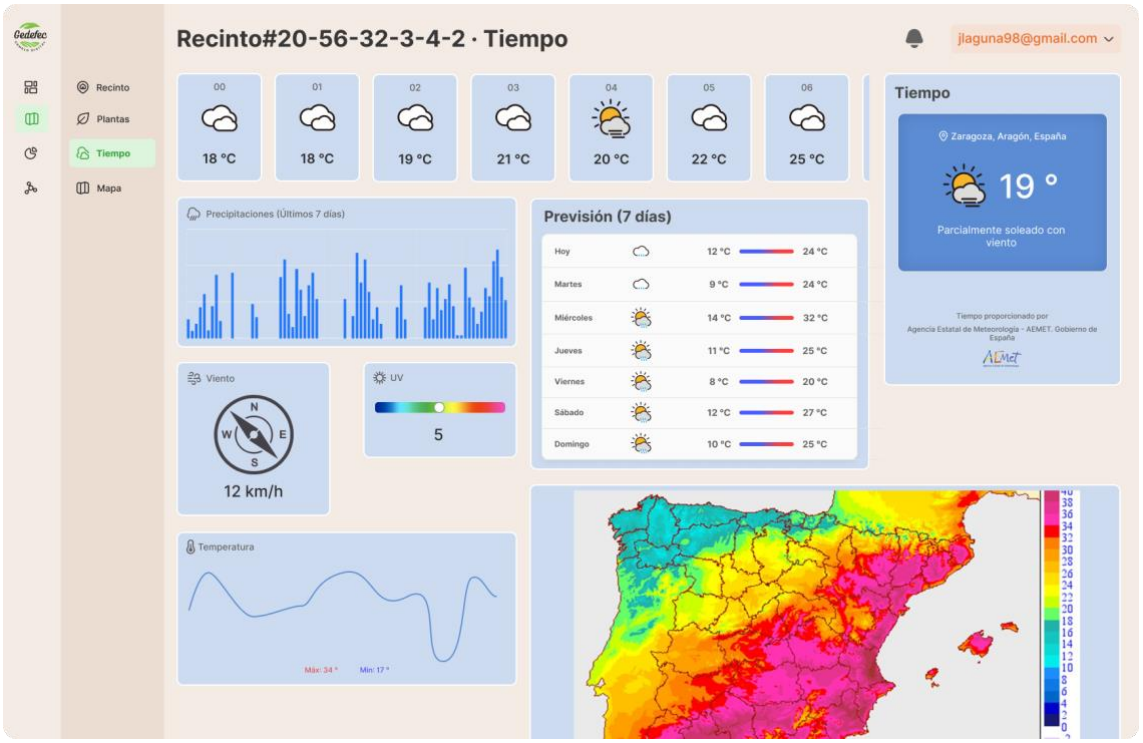


Ilustración 18: Tiempo meteorológico en el recinto

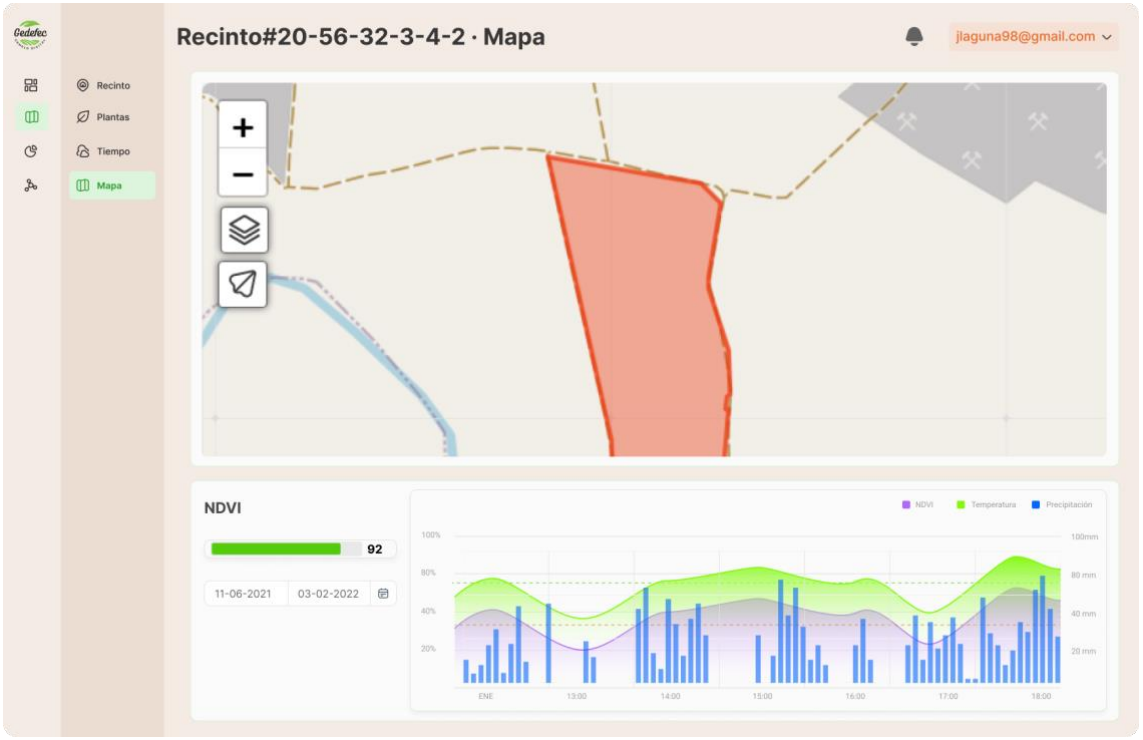


Ilustración 19: Mapa de un recinto. Visualización de mapas de NDVI con más claridad, comparando su valor medio con otras medidas como las precipitaciones o la temperatura

Esta nueva interfaz supone un diseño más uniforme, compacto y bien dividido en distintas partes, en lugar de intentar embeber todo en tres pantallas. Los colores son más coherentes entre sí y tienen sentido. Cada página contiene un *grid* más complejo pero más fácil de leer por el usuario. Obviamente, este diseño de interfaz cambiará de nuevo, modificándose o añadiendo más funcionalidades.

Arquitectura interna de los componentes

Es interesante ver cómo se ha realizado el sistema a nivel de código, tanto en la parte de *backend* como en la parte de *frontend*. En el listado 1 se puede ver cómo está hecha la organización de las carpetas del proyecto en esta primera fase. Hemos tratado de realizar código limpio, modular, escalable, mantenible en el tiempo y testeable. Esto ha sido posible siguiendo los principios de las arquitecturas limpias [22], que facilitan mucho esta tarea, añadiendo, sin embargo, un extra de complejidad y más *boilerplate code*.

[22] https://www.techtarget.com/searchapparchitecture/tip/A-primer-on-the-clean-architecture-pattern-and-its-principles

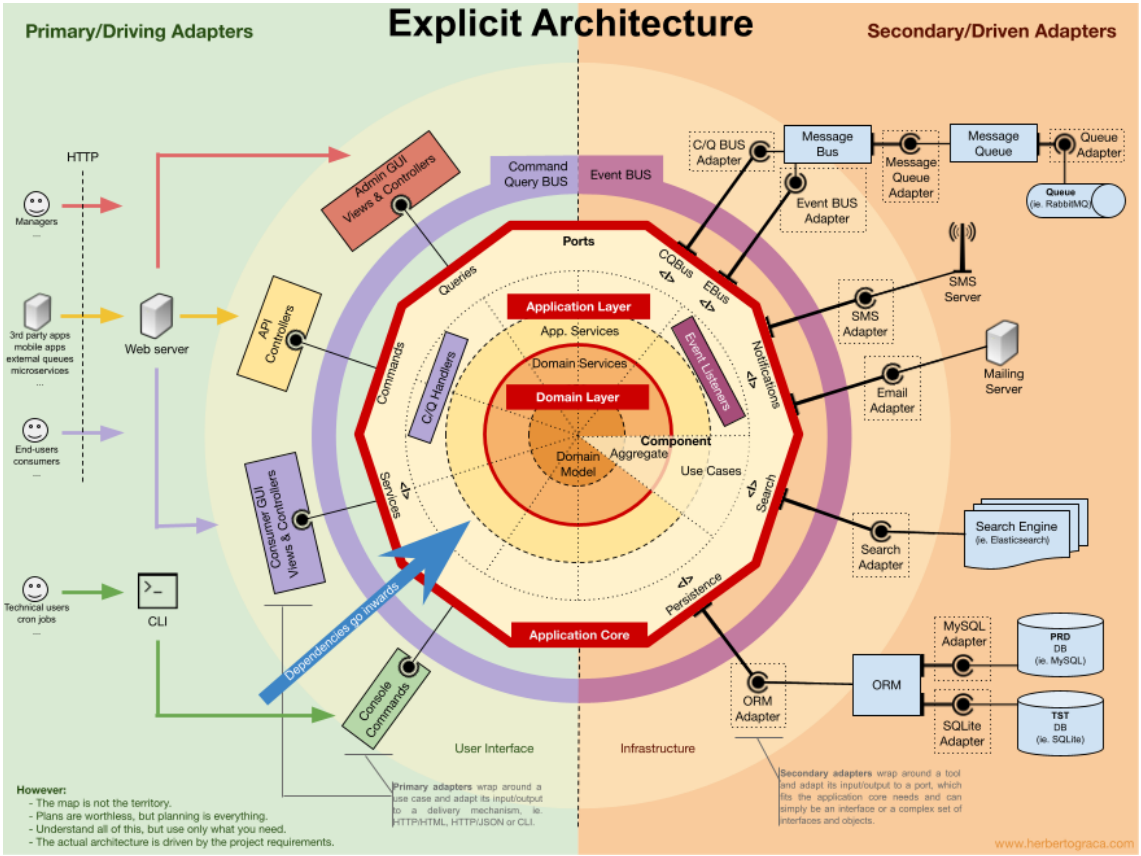
```
.
├── LICENSE
├── README.md
├── backend
│   ├── digital-twin
│   │   ├── data-mart
│   │   ├── etl
│   │   └── main-server
│   ├── esb
│   ├── global-data-management
│   │   ├── batch
│   │   ├── resources
│   │   └── stream
│   └── monitorization
│       ├── grafana
│       ├── prometheus
│       └── secrets
├── docker-compose.dev.yml
├── docker-compose.prod.yml
├── docker-compose.yml
├── frontend
│   ├── Dockerfile
│   ├── digital-twin-panel
│   ├── nginx
│   │   ├── dev.nginx.conf
│   │   └── nginx.conf
└── init-letsencrypt.sh
```

Listado 1: Estructura de carpetas del proyecto.

¿Realmente merece la pena su uso? La respuesta es sí. Las desventajas merecen la pena si se trata de un proyecto que es más complejo que un simple cliente servidor. Un

servidor puede comunicarse con muchas tecnologías y pueden llegar a cambiar rápidamente al inicio de un proyecto. Cambiar esto puede ser un infierno si no se diseña una arquitectura mantenible como esta (esto podría pasar en la arquitectura MVC[23] tradicional). Para este proyecto, usar una arquitectura limpia es, simplemente, incuestionable.

La arquitectura hexagonal[24] es la elegida de entre las arquitecturas limpias, por su forma concisa de separar el dominio de los actores externos con los que interactúa, centrándose en el *Domain-driven design (DDD)*. Se basa en una diseño de dominio, servicios, puertos y adaptadores que podemos ver en la **Error! Reference source not found.**. En resumen, se tiene un dominio con los modelos del negocio. Los servicios interactúan con el dominio y con los puertos establecidos, que son las interfaces que dicen cómo los actores externos deben interactuar con el dominio. Los adaptadores, como su nombre indica, adaptan las interfaces de los puertos para que sean usadas por los actores externos.



[23] Modelo-Vista-Controlador. Arquitectura que se ha usado tradicionalmente en la mayoría de aplicaciones web.

[24] https://herbertograca.com/2017/11/16/explicit-architecture-01-ddd-hexagonal-onion-clean-cqrs-how-i-put-it-all-together/

Ilustración 20: Diseño de la arquitectura hexagonal. Fuente: https://herbertograca.com/2017/11/16/explicit-architecture-01-ddd-hexagonal-onion-clean-cqrs-how-i-put-it-all-together/.

Centrándonos en la Ilustración 3, hemos aplicado variaciones de arquitectura hexagonal en el servidor principal de gin-gonic, en el servicio de Apache Camel, en el servicio de ETL de Prefect e incluso en la aplicación web de Svelte.

El servidor de gin-gonic es el componente que más se aproxima a la arquitectura hexagonal pura, sin modificaciones propias. En el listado 2 se puede ver cómo está la estructura de carpetas en esta primera fase. Perfectamente se pueden ver las partes principales de la arquitectura hexagonal: el dominio, los servicios, los puertos y los adaptadores.

```
.
├── adapters
│   ├── primary
│   │   ├── cli
│   │   ├── events
│   │   │   └── event-handler
│   │   └── web
│   │       └── rest-api
│   └── secondary
│       ├── cache
│       │   └── redis
│       ├── esb
│       │   └── rabbitmq
│       ├── file-storage
│       │   └── local-file-storage
│       └── persistence
│           └── mongodb
├── cmd
│   ├── images
│   │   ├── news
│   │   └── nuts
│   ├── main
│   └── main.go
└── core
    ├── domain
    │   ├── domain-internal.go
    │   └── domain.go
    ├── ports
    │   ├── repositories.go
    │   └── services.go
    └── services
        ├── auth-srv
        │   └── auth-srv.go
        ├── cache-srv
        │   └── cache-srv.go
```

```
    ├── encryption-srv
    │   └── encryption-srv.go
    ├── image-srv
    │   └── image-srv.go
    ├── news-srv
    │   └── news-srv.go
    ├── parcels-srv
    │   └── parcels-srv.go
    └── users-srv
        └── users-srv.go
```

Listado 2: Estructura de carpetas del servidor principal de gin-gonic.

Lo mismo sucede con el

```
.
├── App.svelte
├── app
│   ├── config
│   │   ├── config.ts
│   │   ├── constants.ts
│   │   ├── routes
│   │   └── stores
│   └── styles
│       └── scss
│           ├── abstracts
│           ├── core
│           └── settings
├── assets
├── globals.scss
├── lib
│   ├── core
│   │   ├── Domain.ts
│   │   ├── Exceptions.ts
│   │   ├── functions.ts
│   │   ├── ports
│   │   │   ├── Repository.ts
│   │   │   └── Services.ts
│   │   └── services
│   │       ├── parcel.service.ts
│   │       └── user.service.ts
│   └── infraestructure
│       ├── presentation
│       │   ├── components
│       │   ├── layouts
│       │   └── pages
│       └── repositories
│           ├── http
```

```
|       └── storage.repository.ts
├── main.ts
```

Listado 3: Estructura de ficheros de la aplicación web de Svelte.

En cuanto a Apache Camel, como se ha comentado en la sección 2.2.4, su lenguaje propio en sí hace que su diseño sea, de forma intrínseca, una arquitectura por capas, debido a la alta abstracción del framework. El flujo de información va de la siguiente forma en Camel:

Conector → adaptador → procesamiento (dominio) → adaptador → conector

Lo mismo sucede con los *pipelines* de prefect, dibujando el flujo de información como:

Extract → adaptador → *Transform* (dominio) → adaptador → *Load*

Algo muy importante en el ciclo de vida de un proyecto tanto pequeño como grande, en distinta medida, es el uso del *testing* como herramienta indispensable para mantener y añadir nuevo código en proyectos sin que nada se rompa. En este caso, la testabilidad del código se aumenta considerablemente, debido a la modularidad del código, a la inyección de dependencias y al agnosticismo que eso significa con respecto a las tecnologías y sistemas con los que interactúa el dominio.

La arquitectura hexagonal nos permite introducir inyección de dependencias en el código fácilmente, dividiendo las responsabilidades de los programas diseñados, empezando por los servicios y siguiendo por los repositorios. Por ejemplo, tomamos un ejemplo del programa principal de gin-gonic, donde se inyectan las dependencias de todos los servicios usados.

```
 mongoUri := os.Getenv("MONGO_URI")

 rabbitMQURI := os.Getenv("RABBITMQ_URI")

 mongoDb := os.Getenv("MONGO_DB")

mongodbRepository := mongodb.NewMongodbConn(mongoUri,
mongoDb, 10)

 rabbitMQESB := rabbitmq.NewRabbitMQConn(rabbitMQURI)


 usersService := userssrv.New(mongodbRepository)

 usersHandler := usershdl.NewHTTPHandler(usersService)


 parcelsService := parcelssrv.New(mongodbRepository,
rabbitMQESB)

 parcelsHandler := parcelshdl.NewHTTPHandler(parcelsService)
```

Listado 5: Inyección de dependencias en el servidor de gin-gonic

De esta forma, el código se puede testear fácilmente y hacer *stubs* y/o *mocks* de las dependencias inyectadas para probar todos los casos sin tener que realizar código nuevo solamente para testear los componentes.

Seguridad en el prototipo

En cuanto a la seguridad que tenemos en el prototipo, no se ha hablado mucho en el cuerpo principal debido a que, sin dejar de ser importante, no era el concepto a explicar en este primer prototipo. Sin embargo, existe una mínima seguridad en el prototipo que permite realizar un despliegue a producción temprano y sin problemas.

Autentificación

Como se trata de una primera versión, el primer prototipo tiene una seguridad relativamente sencilla, fuera del uso de software de terceros como Auth0 para autenticación. De momento, se usa un sistema de *HASH + SALT* para autenticación con email y contraseña. El administrador es el que crea las cuentas y le transfiere la información al usuario que la requiera. Dicha información se guarda en la base de datos local del GD.

Autorización

Para la autorización de recursos, se usa un token JWT. Los secretos de este token son conocidos tanto por el *frontend* como por el servidor principal del gemelo digital, que funciona como puerta a los servicios del sistema. Simplemente, el usuario inicia sesión con su email y contraseña y, si se valida correctamente, se entrega al cliente tanto el token JWT de refresco (a través de cookies HTTP-only) y otro token JWT de acceso a recursos, poco duradero en el tiempo, a través de la respuesta HTTP JSON.

Dichos tokens son firmados digitalmente por el servidor, de forma que, si son modificados de alguna forma, se convierten en inválidos. Los datos que se envían son: id de usuario, rol y email.

Esta información no se cifra debido a que tampoco información sensible sobre la que asegurar confidencialidad.

En un futuro, si el proyecto escala a una mayor cantidad de usuarios, el movimiento más inteligente sería buscar un servicio de autenticación y autorización de terceros como lo es Auth0, facilitando esta parte y escalando más fácilmente.

Encriptación

Para reforzar la seguridad de punta a punta entre el *frontend* y el *backend*, se realiza una encriptación de algunos de los datos que se envían (los que se requiere la

confidencialidad) con encriptación de clave simétrica de bloques, usando **AES-CBC**. De momento, más que suficiente junto a TLS para asegurirar la confidencialidad y secretismo de los datos.

Seguridad interna del backend

RabbitMQ va a ser el bróker que haga la comunicación entre los componentes del *backend*, y tiene ya por defecto seguridad, como el uso de usuario y contraseña para autentificar o el usor de *vhosts*[25].

Existe una situación excepcional, en la que, si los consumidores de ciertas colas procesan más lentos que los productores ponen información en dichas colas, puede llegar a haber *overflows* de colas. RabbitMQ tiene distintas estrategias para solucionar esto y que se tendrá que elegir por el programador[26].

Además, existe el caso donde, por cualquier error, la información que se envía no llega al consumidor o existe algún problema en las colas. En este caso, el mensaje fallido se redirige a una cola especial, *Dead letter queue*, que se puede consumir y elegir dos estrategias: poner de vuelta en la cola o eliminar el mensaje definitivamente. Con esto se puede llegar un log de los mensajes que fallan para su posterior investigación u otras estrategias sobre errores.

Convenciones usadas

Algunas de las convenciones que se han usado a la hora de realizar las distintas partes del trabajo son:

- Uso del estándar GeoJSON para las estructuras de datos geográficos que tenemos en el modelo de datos, que son las parcelas y los recintos. Estos son usados para representarlos gráficamente en los mapas de *leaflet* y poder hacer operaciones como el cálculo de distancia entre parcelas de forma sencilla.
- Así como se ha usado la convención geográfica, también se usan muchos datos históricos, los cuales se tienen que usar para alimentar a los gemelos digitales. Por lo tanto, muchas entidades de la base de datos tienen asociados un *timestamp,* además de separarlos en colecciones para mayor flexibilidad y rapidez en la base de datos. MongoDB tiene colecciones hechas para este tipo de usos[27].
- Uso de una estructura de datos para emular la etiquetación con metadatos en un *data lake*. En este caso, con base de datos de MongoDB, los metadatos que se usan son:

[25] Más información sobre los *vhosts* de rabbit: https://www.rabbitmq.com/vhosts.html

[26] Más información sobre los overflow de colas: https://www.rabbitmq.com/maxlength.html

[27] https://www.mongodb.com/docs/manual/core/timeseries-collections/#time-series-collections

```
meta {
        author: string;
        traceability: List<string>;
        created_at: timestamp;
        tags: List<string>;
        type: string;
        category: string;
}
```