IoT Aplicado a la Agricultura y Ganadería (IOTA2F)

JAIME DE LA PEÑA RAMOS

FACULTAD DE CIENCIAS

UNIVERSIDAD DE SALAMANCA

TRABAJO FIN DE GRADO EN INGENIERÍA INFORMÁTICA

JULIO DE 2021

Tutores: Dr. Fernando de la Prieta Pintado y Dra. Elena Pascual Corral

Agradecimientos

Quiero empezar agradeciendo a mis tutores la oportunidad de realizar este proyecto y su gran esfuerzo y dedicación en la supervisión de este.

Agradecer también a mi familia, a mis padres, a mis hermanos, a quienes ya no están, me habéis hecho crecer como persona y demostrar que todo es posible de conseguir con esfuerzo y dedicación.

Y una especial mención a mi pareja, que siempre ha estado a mi lado apoyándome y animándome, para que diera lo mejor.

Sin más que añadir, gracias por todo.

Dr. Fernando de la Prieta Pintado, personal del Departamento de Informática y Automática y Dra. Elena Pascual Corral, personal del Departamento de Física Aplicada. Ambos de la Universidad de Salamanca.

CERTIFICAN:

Que el trabajo titulado “IoT Aplicado a la Agricultura y Ganadería (IOTA2F)” ha sido realizado por D. Jaime de la Peña Ramos, con DNI 70921486C, para la asignatura Trabajo de Fin de Grado de la titulación Grado en Ingeniería Informática de la Universidad de Salamanca.

Y para que así conste a todos los efectos oportunos.

En Salamanca, a 4 de julio de 2021

D. JAIME DE LA PEÑA RAMOS

DE LA PEÑA RAMOS JAIME - 70921486C

Firmado digitalmente por DE LA PEÑA RAMOS JAIME - 70921486C
Fecha: 2021.07.04 13:28:11 +02'00'

Dr. FERNANDO DE LA PRIETA PINTADO

DE LA PRIETA PINTADO FERNANDO - 71011007W

Firmado digitalmente por DE LA PRIETA PINTADO FERNANDO - 71011007W
Fecha: 2021.07.05 18:34:24 +02'00'

Dra. ELENA PASCUAL CORRAL

PASCUAL CORRAL ELENA - DNI 44402830L

Firmado digitalmente por PASCUAL CORRAL ELENA - DNI 44402830L
Fecha: 2021.07.05 10:56:47 +02'00'

Resumen

El presente documento tiene como propósito la explicación del desarrollo de un sistema de monitorización sobre diferentes condiciones que intervienen en un entorno agrícola-ganadero, y la gestión y control de diferentes usuarios que pueden hacer uso de este.

La interfaz web ha sido diseñada e implementada para ser utilizada por un amplio rango de usuarios, con mayor o menor conocimiento de las TIC (Tecnologías de la Información y la Comunicación), ya que cuenta con una semántica y apariencia sencilla y amigable. Además, ha sido desarrollada para que sea responsiva en la mayoría de los dispositivos como *smartphones*, *tablets*, ordenadores, etc.

La funcionalidad del sistema engloba, desde la monitorización de diferentes condiciones ambientales, como temperatura, humedad, precipitaciones, monitorización de sistemas de riegos, sistemas de detección incendios, localización GPS y categorización del ganado bovino, hasta el control y gestión de usuarios, y exportación de los datos.

La tecnología utilizada permite la ejecución del código *on-server*. De esta forma, se puede hacer uso del sistema únicamente a través de un navegador web. Esta tecnología ha sido desarrollada a través de un *framework* de PHP llamado Yii2, el cual permite a su vez la utilización de otros, como JQuery (para el código JavaScript) y Bootstrap 3 (para el código HTML y CSS). La arquitectura de este *framework*, basado en componentes y compatibilidad de caché, lo hace muy apropiado para el desarrollo de aplicaciones web.

Además, se ha decidido utilizar un *backend on-cloud* a través del proveedor de *cloud* AWS (Amazon Web Services), el cual, ofrece infraestructuras, bases de datos, memorias caché, almacenamiento compartido en forma de servicios, lo que brinda al sistema de una alta disponibilidad y una gran tolerancia a fallos, garantizando un SLA (*Service Level Agreement*) del 99.99% al mes en el servicio de infraestructura (EC2) y del 99.95% al mes en el servicio de base de datos (RDS).

De esta manera, finalmente, se ha podido comprobar que la utilización de este sistema supone un incremento en el rendimiento de la producción agrícola y un ahorro significativo en cuanto al coste económico.

Abstract

The following document has as purpose to explain the development of a monitoring system over different conditions that intervene in an agricultural-farming level, and the management and control different users can make out of it.

The web interface has been designed and implemented to be used by a wide range of users, with greater or lesser knowledge of ICTs (Information and Communication Technologies), as it offers a friendly and straightforward appearance and semantics. Furthermore, it has been developed to be responsive in the wide majority of devices such as smartphones, tablets, computers etc.

The functionality of the system includes, from monitoring of different environmental conditions like temperature, humidity, precipitation, risk system monitoring, fire-detection systems, GPS and livestock sorting of cattle, to user control and management, and data export.

The technology used allows the execution of the *on-server* code. This way, the system can only be accessed through a web navigator. This technology has been developed through a PHP framework called Yii2, which allows the use of others, such as JQuery (for JavaScript code) and Bootstrap 3 (for HTML and CSS code). By virtue of this framework's architecture based in cache components and compatibility, results very appropriate for the development of web applications.

In addition, the system uses a *backend on-cloud* through the AWS (Amazon Web Services) provider, which offers infrastructure, databases, cache memory, and shared storage in the form of services. This provides the system great availability and fault tolerance, to guarantee a 99.99% SLA (Service Level Agreement) per month in the service infrastructure, and a 99.95% per month in the database service (RDS).

Thus, finally, the use of this system has been proved to imply an increase in agricultural production performance and a significant saving in terms of economic costs.

Tabla de contenido

Resumen 6

Abstract 7

Índice de figuras 11

Índice de tablas 14

Listado de acrónimos 15

1. Introducción 17

2. Objetivos 21

2.1. Objetivo principal del sistema 21

2.2 Objetivos personales del proyecto 22

3. Aspectos teóricos 23

3.1. Cloud 23

3.1.1. Arquitectura en la nube 23

3.1.2. Comparación entorno cloud VS On-Premise 25

3.1.3. Amazon Web Services 27

3.3. Internet of the Things 31

4. Técnicas y Herramientas 34

4.1. Aplicación web y *frameworks* de desarrollo 34

4.2. Herramientas utilizadas 36

5. Aspectos relevantes 42

5.1. Seguimiento del sistema 42

5.2. Metodología 45

5.1.1. Metodología de trabajo 45

5.1.2. Estimación del esfuerzo y planificación temporal 51

5.1.3. Especificación de requisitos 53

5.1.4. Análisis del sistema 55

5.1.5. Diseño del sistema 56

5.3. Componentes Software 58
5.4. Componentes Hardware 60
5.5. Despliegue en AWS 63
6. Conclusiones 71
7. Líneas de trabajo futuras 73
8. Bibliografía 75

Índice de figuras

Figura 1. Tamaño del mercado agrícola mundial del IoT en 2018 y 2023 (en miles de millones de dólares estadounidenses). Statista (2021) 18
Figura 2. Cuadrante Mágico de Gartner que refleja el crecimiento de los proveedores cloud. Gartner (2020) 27
Figura 3. Porcentaje del gasto mundial en infraestructura cloud. Canalys (2020) 28
Figura 4. Alcance de la red de AWS en el mundo. Amazon Web Services (2021) 30
Figura 5. Topología protocolo MQTT. HWlibre (2020) 32
Figura 6. Modularidad de aplicación en Yii2 y relaciones existentes. Elaboración Propia 35
Figura 7. Primera versión del sistema con PHP sin framework de desarrollo. Elaboración Propia 42
Figura 8. Tercera versión del sistema con framework de desarrollo y dashboard. Elaboración Propia 43
Figura 9. Versión final del sistema. Elaboración Propia 44
Figura 10. Diagrama de arquitectura entorno demo. Elaboración Propia 45
Figura 11. Tablero utilizado en el método Kanban para la elaboración del sistema. Elaboración Propia 46
Figura 12. Hito de la primera versión del sistema sin tema. Elaboración Propia 49
Figura 13. Código desplegado en el repositorio de GitLab y estructura en ramas. Elaboración Propia 50
Figura 14. Estimación del esfuerzo para el desarrollo del sistema en la herramienta EZEstimate. Elaboración Propia 51
Figura 15. Planificación temporal para el desarrollo del sistema en la herramienta Tom's Planner. Elaboración Propia 52
Figura 16. Diagrama de paquetes del sistema. Elaboración Propia 55
Figura 17. Patrón MVC empleado en el sistema representado a través de la clase User. Elaboración Propia 56
Figura 18. Patrón Composite empleado en el sistema. Elaboración Propia 57
Figura 19. Patrón Abstract Factory empleado en el sistema. Elaboración Propia 57
Figura 20. Patrón DAO empleado en el sistema para acceder a la BDD. Elaboración Propia 58
Figura 21. Diagrama de despliegue del sistema. Elaboración Propia 58

Figura 22. Circuitería del prototipo hardware desarrollado. Elaboración Propia..... 60
Figura 23. Vista principal del prototipo implementado. Elaboración Propia..... 61
Figura 24. Vista interior de prototipo implementado. Elaboración Propia..... 62
Figura 25. Captura de pantalla de la RaspBerry Pi4 con los datos obtenidos del sensor. Elaboración Propia..... 62
Figura 26. Ejecución comando de planificación de Terraform. Elaboración Propia..... 64
Figura 27. Aplicación del comando que aplica el plan de ejecución en Terraform. Elaboración Propia..... 65
Figura 28. Roles principales del playbook de ejecución del Ansible. Elaboración Propia 65
Figura 29. Fin de la provisión del Ansible. Elaboración Propia 66
Figura 30. Diagrama de arquitectura del sistema en el entorno pre-productivo. Elaboración Propia..... 66
Figura 31. Diagrama de arquitectura del sistema en el entorno productivo. Elaboración Propia 68
Figura 32. Proceso de aplicación de parches y políticas de seguridad automático en AWS. Elaboración Propia..... 69
Figura 33. Proceso de creación de AMI y actualización del ASG de forma automática en AWS. Elaboración Propia 70
Figura 34. Estimación temporal para las nuevas integraciones. Elaboración propia 73
Figura 35. Diagrama de Gantt para las nuevas integraciones. Elaboración propia..... 74

Índice de tablas

Tabla 1. Comparación de las características de los proveedores cloud AWS, Microsoft Azure y GCP. Elaboración Propia 28
Tabla 2. Comparación de diferentes tecnologías de IoT. Elaboración Propia 33
Tabla 3. Sprints seguidos en la realización del sistema. Elaboración Propia..... 47
Tabla 4. Ejemplo de plantilla, a partir del caso de uso UC-0001, para la recolección de requisitos a través del método de Durán y Bernárdez. Elaboración Propia 53

Listado de acrónimos

Acrónimo	Significado
AMI	Amazon Machine Image
API	Application Programming Interface
ASG	Auto Scaling Group
AWS	Amazon Web Services
AZ	Availability Zone
CPD	Centro de Procesamiento de Datos
CRUD	Create, Read, Update, Delete
DAO	Data Access Object
EC	ElastiCache
EC2	Elastic Cloud Compute
ECS	Elastic Container Service
EFS	Elastic File System
ELB	Elastic Load Balancer
GCP	Google Cloud Platform
GPS	Global Positioning System
HTML	HyperText Markup Language
I2C	Inter-Integrated Circuit
IaaS	Infraestructure as a Service
IAM	Identity Access Management
IFTTT	IF This, Then That
IoT	Internet of the Things
IP	Internet Protocol
IPS	Instrucciones Por Segundo
JS	JavaScript
LCD	Liquid-Crystal Display
M2M	Machine to Machine
MQTT	Message Queue Telemetry Transport
MQTTS	Message Queue Telemetry Transport Secure
MVC	Modelo Vista Controlador
NACL	Network Access Control List

ONUAA	Organización de las Naciones Unidas para la Alimentación y la Agricultura
PaaS	Platform as a Service
PHP	Hypertext Preprocessor
PU	Proceso Unificado
PUA	Proceso Unificado Agil
QoS	Quality of Service
RDS	Relational Database Service
S3	Simple Storage Service
SaaS	Software as a Service
SAP	Systems, Applications, Products in Data Processing
SARS-CoV-2	coronavirus de tipo 2 causante del síndrome respiratorio agudo severo
SCL	System Clock
SDA	System Data
SES	Simple Email Service
SG	Security Group
SN	SubNet
SNS	Simple Notification Service
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TI	Tecnologías de la Información
TIC	Tecnologías de la Información y Comunicación
TLS	Transport Layer Security
UCP	Use Case Points
UI	User Interface
UX	User eXperience
VPC	Virtual Private Cloud
VPG	Virtual Private Gateway
Yii	Yes, it is

1. Introducción

La sobrepoblación es un riesgo de un futuro inminente. La Organización de las Naciones Unidas para la Alimentación y la Agricultura (ONUAA) o más conocida como FAO, afirma que el mundo necesitará producir un 70% más de alimentos en 2050 (FAO, 2009). La disponibilidad limitada de recursos naturales, como agua dulce o tierras cultivables, junto con las escasas tendencias de rendimiento en los cultivos básicos, ha acelerado aún más este problema.

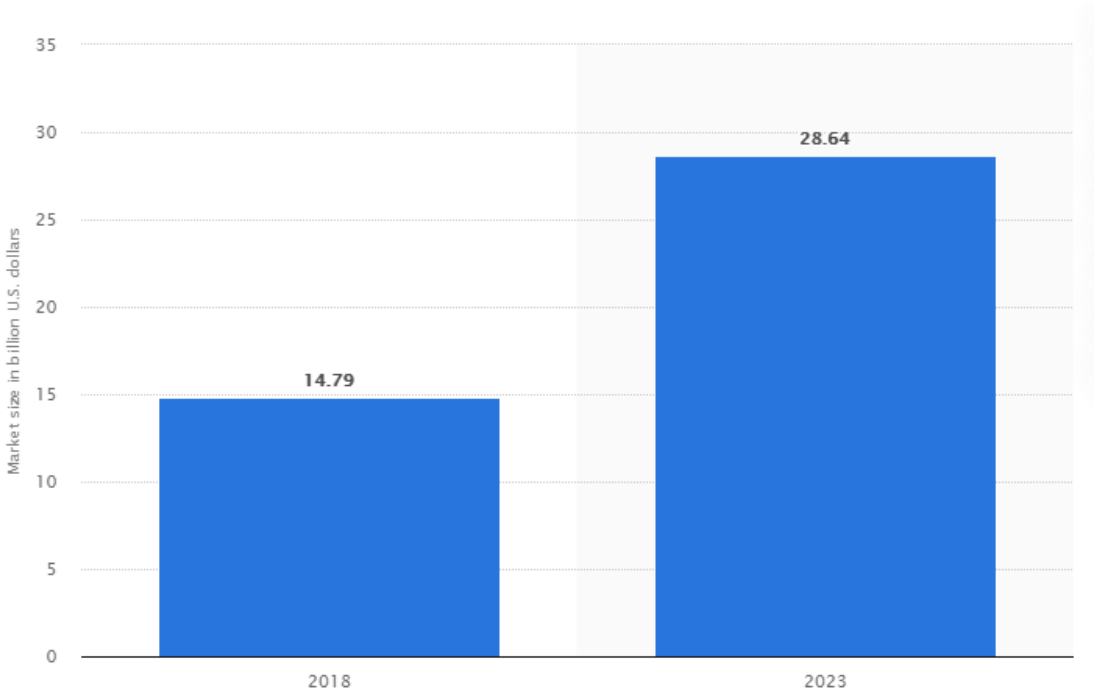
Otra preocupación es la mano de obra agrícola, ya que ha disminuido en la mayoría de los países y como consecuencia de esta disminución se ha aumentado el uso de soluciones con conectividad a Internet con objetivo de reducir la necesidad de los trabajos manuales. Estas tecnologías son muy variadas, ya que pueden ir desde el uso de drones para el monitoreo de cultivos, ya que estos dispositivos permiten proporcionar una vista aérea del terreno. O utilizar sensores para la recogida de datos relativos a los terrenos de cultivo, de forma que estos permitan analizar y enviar información sobre este, conocer la temperatura, humedad del suelo, estado de los riegos, etc (IAT, 2020). Para así tener una monitorización sobre estos y por ejemplo, saber cuándo es el momento óptimo para aplicar sulfitos u otros minerales en el terreno.

Las soluciones de IoT (*Internet of the Things*) pretenden ayudar a los agricultores a reducir la brecha entre la oferta y la demanda, garantizando una mayor eficiencia, rentabilidad, sostenibilidad y protección al medio ambiente. El enfoque del uso de esta tecnología es garantizar un empleo óptimo de los recursos para conseguir altos **rendimientos** de los cultivos y reducir los costes.

Se estima que el mercado global de IoT en la agricultura crecerá con una tasa compuesta anual notable del 14.7 % entre los años 2018 y 2025, debido a la reducción del coste tecnológico y al compromiso de los gobiernos de los países de todo el mundo para aumentar la calidad y cantidad de la producción agrícola (Kadam, 2018).

A continuación, en la **Figura 1**, puede verse un gráfico comparativo sobre el tamaño del mercado a nivel mundial del IoT del año 2018 respecto a lo que se estima en el año 2023.

Figura 1. *Tamaño del mercado agrícola mundial del IoT en 2018 y 2023 (en miles de millones de dólares estadounidenses).* Statista (2021)



A lo largo de la realización de este sistema, se han estudiado diferentes investigaciones, realizadas por otros autores cuyas propuestas han permitido ampliar el conocimiento sobre las necesidades ya mencionadas en el apartado de introducción, y qué mejoras han planteado en cuanto al desarrollo agrícola, las cuales se desarrollan a continuación.

Según Nikesh Gondchawar y Prof. Dr. R. S. Kawitkar (2016) a través de tecnologías de automatización e IoT tienen como objetivo modernizar los actuales métodos tradicionales de agricultura, por ello, entre las características más destacadas de su proyecto han elaborado un robot que lleve a cabo tareas como el deshierbe, pulverización, control de humedad, espantar plagas, etc. Gestión de riegos y de almacenes todo esto a través de módulos WiFi y Zigbee y actuadores como microcontroladores y RaspBerry Pi.

Por otro lado, según las investigaciones realizadas por Andreas Kamilaris, Feng Gao, Francesc X. Prenafeta-Boldu y Muhammad Intizar Ali (2016) han propuesto un marco semántico para aplicaciones agrícolas inteligentes basadas en IoT, que permite razonar sobre varios flujos de datos de sensores heterogéneos en tiempo real.

En comparación con soluciones basadas en IoT ya existentes y con motivo de abordar la problemática descrita previamente, en el presente documento se plantea el desarrollo de un sistema que permita monitorizar y tener un registro de aspectos que repercuten en la productividad de un terreno agrícola-ganadero, como pueden ser la temperatura, humedad ambiental, humedad del terreno, estado de los riegos, etc. Además, permitirá llevar una trazabilidad de los proveedores que acceden a la finca de forma presencial, cuya necesidad surge a raíz de la pandemia ocasionada por el SARS-CoV-2 en el año 2020. Y permitir una gestión y control tanto de este tipo de usuarios como de los administradores que puedan acceder al sistema. Finalmente, se podrá realizar una correcta gestión de los archivos y directorios que se desean manejar en el sistema

Así mismo, la estructura seguida en este documento es la que se detalla a continuación.

- **Introducción**: Apartado donde se describe las causas del sistema planteado y la problemática que pretende abordar, también contiene el análisis de otros sistemas similares planteados y las mejoras respecto a estos.
- **Objetivos**: Aquí se detallan los objetivos principales del sistema y los objetivos personales que se quieren conseguir con la elaboración de este.
- **Aspectos teóricos**: Donde se detallan los aspectos clave de las tecnologías *on-cloud*, se describen las ventajas de la utilización del proveedor *cloud* Amazon Web Services y aspectos básicos del IoT y del protocolo M2M implementado.
- **Técnicas y Herramientas**: Se describen las técnicas y herramientas de desarrollo, explicando los motivos de su utilización y finalmente, las herramientas utilizadas y por qué esas.
- **Aspectos Relevantes**: Como su nombre indica, se recogen los puntos clave del sistema, es decir, su desarrollo y evolución, la metodología implementada, los componentes software y hardware que intervienen y el proceso de despliegue en Amazon Web Services.
- **Conclusiones**: Aquí se recogen las conclusiones tras la elaboración del sistema y los objetivos logrados.
- **Líneas de trabajo futuras**: Donde se estudiarán aspectos a mejorar en el sistema y nuevas funcionalidades para añadir, incluyendo además una estimación del tiempo necesaria para esto.
- **Bibliografía**: Último apartado donde se recogen las referencias consultadas y utilizadas en este documento.

Si así desea el lector, este puede profundizar más en el desarrollo de este documento, a través de un contenido más ampliado en diez anexos que se dividen en:

- **Anexo I. Planificación Temporal**: Incluye la planificación de tareas para el desarrollo del proyecto, superpuestas sobre una línea temporal sobre el marco de un calendario.
- **Anexo II. Especificación de Requisitos y Análisis del Sistema**: Conjunto de los requisitos funcionales, no funcionales, de información, que debe cumplir el sistema para alcanzar los objetivos principales y subobjetivos de estos y desarrollo de los casos de uso del sistema a partir de diagramas de secuencia, modelo de dominio y paquetes de análisis.
- **Anexo III. Diseño del Sistema**: Diseño de la arquitectura del sistema, diseño de la interfaz y pruebas. Explicación de los diferentes servicios de AWS utilizados en el proyecto. Definición de la forma, función, utilidad, imagen de marca y otros aspectos que afectan a la navegación del usuario por la aplicación web. Descripción técnica y definición de los dispositivos hardware que intervienen en el sistema y. finalmente, el procedimiento a realizar para desplegar infraestructura en el proveedor *cloud* AWS.
- **Anexo IV. Manual del Programador**: Manual para los programadores, que traten el código y tengan más claro su funcionamiento y estructura.
- **Anexo V. Manual del Usuario**: Manual centrado para la utilización del usuario final, que entienda la funcionalidad del sistema y tenga una forma clara de cómo utilizarlo de forma correcta.

2. Objetivos

El desarrollo de este sistema tiene como fin la superación de una serie de objetivos tanto a nivel técnico como a nivel personal, estos son los siguientes.

2.1. Objetivo principal del sistema

El objetivo principal con la realización de este sistema es tener una mayor gestión y control de aspectos que influyen en una explotación agrícola-ganadera para así facilitar la mano de obra del personal de esta y aumentar la productividad de los recursos. Este objetivo se puede dividir en más específicos, los cuales son los siguientes:

- **Gestión Agrícola**: Este objetivo del sistema pretende la gestión de datos referentes a las temperaturas, humedad ambiental, precipitaciones, estado de los riegos, etc. Proporcionadas por dispositivos en el sistema, se registrará el histórico diario mostrando la fecha y hora del registro y el dispositivo que ha proporcionado estos datos. Estos podrán ser visualizados en formato de tabla, gráfica o incluso, exportarlos, etc. Dependiendo del dato que se esté registrando, se tienen como subobjetivos los siguientes: Gestión de las temperaturas, gestión de la humedad ambiental, gestión del estado de los riegos, gestión de detección de incendios, gestión de las precipitaciones y gestión de la humedad del terreno.
- **Gestión Ganadera**: Por otro lado, este objetivo procura la gestión del ganado, en este caso bovino, del sistema. Por lo que se podrá registrar el ganado que se vaya adquiriendo con datos descriptivos como la raza, género o si es para criar o vender. Se podrá geolocalizar a este ganado y se podrá visualizar en un mapa donde se encuentran cada una de ellas. Existe la posibilidad de editar y eliminar las unidades de ganado bovino que existan en el sistema en caso de ser necesario.
- **Gestión de Recursos Humanos**: Que tiene los siguientes subobjetivos la gestión de proveedores y administradores, quienes son los roles fundamentales que accederán al sistema. Dependiendo del rol, se tendrá un control sobre los datos que atañen al sistema para la visualización de las temperaturas, humedad, o incluso control de los proveedores. Los proveedores podrán obtener sus datos registrados en el sistema y consultar sus visitas físicas.
- **Gestión del Sistema**: Finalmente, este objetivo pretende la correcta sincronización del sistema con los datos y el correcto manejo de estos. Y la gestión de archivos y directorios que se manejen en este.

2.2 Objetivos personales del proyecto

El propósito personal con la elaboración de este proyecto es la adquisición de nuevas habilidades y conocimientos en tecnologías novedosas.

Para el desarrollo del ***frontend*** del sistema, se ha utilizado un *framework* de desarrollo de software, recorriendo etapas de aprendizaje, manejo y correcto desempeño. Por otro lado, se han asimilado nuevos lenguajes de programación como son el PHP y YML entre otros.

También se han implementado tecnologías para el despliegue del ***backend*** de la infraestructura *on-cloud,* las cuales, hoy día son utilizadas por la versatilidad y ventajas que ofrecen como la escalabilidad o redundancia entre otras. El despliegue de microservicios a través de imágenes de Docker en contenedores autogestionados por el proveedor cloud. Configuración de un sistema IoT a través del protocolo MQTTS (*Message Queue Telemetry Transport Secure*) y las ventajas que este ofrece.

Se ha procurado utilizar las mejores prácticas en el diseño de la interfaz, haciendo uso de metodologías propias del **UX** (*User eXperience*) analizando aspectos claves como la usabilidad, la **UI** (*User Interface*), la navegación, los *copys,* la retroalimentación del sistema.

Para la **gestión** del proyecto, se han utilizado la metodología de Proceso Unificado Ágil (PUA), esta variante más actualizada y moderna del Proceso Unificado (PU) permite una mayor resiliencia a cambios en los requisitos del sistema, facilitando el proceso de desarrollo y permitiendo que sea más colaborativo, lo que se ve reflejado en una mayor calidad de producto.

Finalmente, se ha logrado adoptar los diferentes roles presentes día a día en el mundo de las TI desarrollando mis capacidades como profesional permitiéndome crecer laboralmente y empatizar con el resto de los equipos en la elaboración de proyectos.

3. Aspectos teóricos

En este apartado, se va a exponer los apartados teóricos de las soluciones implementadas, en primer lugar, describir y asimilar los conceptos básicos de un entorno *cloud*, qué lo componen, diferencias respecto a un entorno clásico y los principales proveedores *cloud*.

Se hará una descripción del proveedor elegido y cuáles son los beneficios de su utilización y finalmente, se describirá el protocolo *Machine to Machine* (M2M) implementado, las ventajas de este y una comparación respecto a otros.

3.1. Cloud

Antes de comenzar, cabe destacar que se va a utilizar el término *cloud* para referirse a "la nube" a lo largo del documento. Según Red Hat (2021) *cloud* es un término que se refiere a "entornos de TI que extraen, agrupan y comparten recursos escalables en una red. Suelen crearse para habilitar *el cloud computing*, que consiste en ejecutar cargas de trabajo dentro del sistema". No obstante, es importante diferenciar el término *cloud* de *cloud computing*.

- ***Cloud***: Es un entorno, el lugar donde se ejecutan las aplicaciones.
- ***Cloud Computing***: Es una acción, es el proceso de ejecución de cargas de trabajo en la nube.

3.1.1. Arquitectura en la nube

La **arquitectura *on cloud*** constituye la forma en la que se integran las distintas tecnologías para crear los entornos *cloud*.

Podría realizarse un símil con la construcción de una vivienda: la infraestructura de *cloud* incorpora los materiales (almacenamiento, *hardware*, virtualización, red) mientras que la arquitectura *on cloud* diseña y construye la vivienda.

En función del tipo de la plataforma *cloud*, según IBM (2021), se puede hacer una distinción en tres tipos de capas, las cuales son:

1. **Software como servicio (SaaS)**: Un proveedor de servicios proporciona el software, los usuarios se suscriben a este y acceden a él a través de la web o las APIs del proveedor. Los proveedores son los encargados de la disponibilidad y funcionalidad de los servicios. Este es el caso del servicio de correo de Gmail.

2. **Plataforma como servicio (PaaS)**: Un proveedor ofrece acceso a un entorno cloud, en el cual los usuarios pueden crear y distribuir aplicaciones. El proveedor proporciona la infraestructura y es el encargado de mantener la seguridad y escalabilidad de los recursos. Este es el caso de Amazon Web Services o Azure.
3. **Infraestructura como servicio (IaaS)**: Un proveedor proporciona a los clientes acceso gratuito o de pago por el uso de almacenamiento, redes, servidores y otros recursos. El proveedor es el encargado de una correcta escalabilidad automática o semi automática. Este es el caso de Dropbox o One Drive.

Las nubes se consideran entornos de plataformas como servicio (PaaS), ya que un proveedor ofrece a los usuarios la plataforma y la infraestructura subyacente. En función del acceso a estas, según Azure (2021), se puede diferenciar en distintos tipos de *clouds*:

- **Arquitectura de nube pública**: El tipo más común de implementación. Los recursos en la nube (servidores, almacenamiento, etc.) son propiedad de un proveedor de servicios en la nube que los administra y ofrece a través de Internet. Con este tipo de nube, el hardware, software y demás componentes de la infraestructura son propiedad del proveedor, quien también los administra. Entre las ventajas que aporta se encuentran: costes inferiores, sin mantenimiento, escalabilidad casi ilimitada, gran confidencialidad. Algunos ejemplos son: Amazon Web Services, Microsoft Azure, Google Cloud Platform o Alibaba Cloud.
- **Arquitectura de nube privada**: Está compuesta por recursos informáticos que utiliza exclusivamente una empresa u organización. Esta puede ubicarse en el Centro de Procesamiento de Datos (CPD) local de la organización u hospedarla en un proveedor de servicios externo. En este tipo de nube los servicios y la infraestructura siempre se mantienen en una red privada. Entre las ventajas que aporta se encuentran: mayor flexibilidad, más control, más escalabilidad. Un ejemplo de este tipo de arquitectura es la de OpenStack.
- **Arquitectura de nube híbrida**: Este tipo de nube combina la infraestructura de una nube privada con una nube pública. Permiten que los datos y aplicaciones se muevan entre los dos entornos. Muchas organizaciones eligen este tipo de nube por exigencias del negocio como pueden ser el cumplimiento de requisitos de tipo normativo, aprovechamiento de los recursos, solucionar problemas de latencia, etc. Un ejemplo de este tipo de arquitectura es la de Red Hat OpenShift.

- **Arquitectura multicloud**: Incluyen más de una nube, pública o privada y que pueden estar conectadas en red (o no), es lo que explica RedHat (2021).

3.1.2. Comparación entorno cloud VS On-Premise

Ahora que ya se conoce los diferentes aspectos relacionados con el entorno *cloud*, su definición, infraestructura, arquitectura, tipos. Se va a proceder a diferenciar este tipo de entorno con una infraestructura y arquitectura *On-Premise* (en local).

Cuando se habla de este tipo de entornos, según Linkeit (2021), hay que referirse a una instalación de un sistema en una ubicación física, es decir, un entorno TI en local, por lo que el equipo de TI tiene un control total sobre estos. Por otro lado, estos centros son adaptados a las necesidades específicas del cliente, por lo que pueden ser usados para ejecutar desde servidores privados hasta aquellos por terceras partes. No obstante, existe un incremento en el coste por el mantenimiento de este tipo de infraestructura, ya que es el propio cliente el encargado de la configuración y mantenimiento del hardware, de comprar licencias, actualizaciones del sistema, etc.

Algunas diferencias clave entre estos dos tipos de entornos son:

- **Despliegue**: En una arquitectura *On-Premise*, las tareas de ejecución se implementan internamente, en comparación si los datos son ejecutados en un entorno *cloud*, ya que los proveedores son los responsables de la gestión de las tareas de migración en los centros de datos. Incluso, si el despliegue es realizado en una nube pública, privada o híbrida, el propietario tiene el acceso total y completo a esos recursos cuando lo desee.
- **Mantenimiento**: Debido a que los centros de datos y servidores de una infraestructura *On-Premise* están alojados en instancias locales, son los propios dueños los que tienen que ocuparse del mantenimiento, y dependiendo del tamaño de la infraestructura es posible que exista una necesidad en disponer de un equipo de TI interno que supervise y monitorice la infraestructura. En contraposición de las soluciones *cloud*, que eliminan las tareas relacionadas con este punto, ya que este rol recae en el proveedor de los servicios. Por lo que esto ofrece ventajas tales como una rápida escalabilidad y actualización en función de las demandas del cliente.
- **Costes**: El coste inicial de una instalación *On-Premise* es rápidamente calculado, ya que exige una inversión inicial en *hardware*, *software* y un equipo TI de

expertos. También se debe tener en cuenta los gastos de mantenimiento y copias de seguridad, así como de las actualizaciones del sistema. Por otro lado, un sistema *cloud* es más rentable gracias a los modelos propuestos de *pay-per-use*, de AWS, donde solo se paga por los recursos utilizados.

- **Seguridad**: Cuando se habla de seguridad, con los despliegues *On-Premise*, es posible controlar la seguridad física, mediante controles de presencia, y mitigar así posibles amenazas, de hecho, esta puede ser la mayor ventaja de este entorno. En comparación con las soluciones *cloud* las cuales requieren una confianza una confianza íntegra con el proveedor de servicios. Hoy en día estos proveedores garantizan una mayor seguridad y tiempos de disponibilidad más altos.

Finalmente, con estas diferencias, hay que tener en cuenta las necesidades del entorno en el que se quiera operar para saber qué alternativa es mejor que la otra. Es decir, si se necesita un nivel de control más alto en cuanto a la seguridad de acceso a los recursos *hardware*, un entorno *On-Premise* sería una buena elección. En cambio, si lo que prima es un entorno altamente escalable, a un coste asequible y con un mantenimiento y soporte eficaz, el entorno *cloud* es la mejor elección.

No obstante, siempre existiría la posibilidad de la migración de cargas de trabajo a la nube, este servicio lo ofrece algunos proveedores como es el caso de Amazon Web Services con los despliegues SAP.

3.1.3. Amazon Web Services

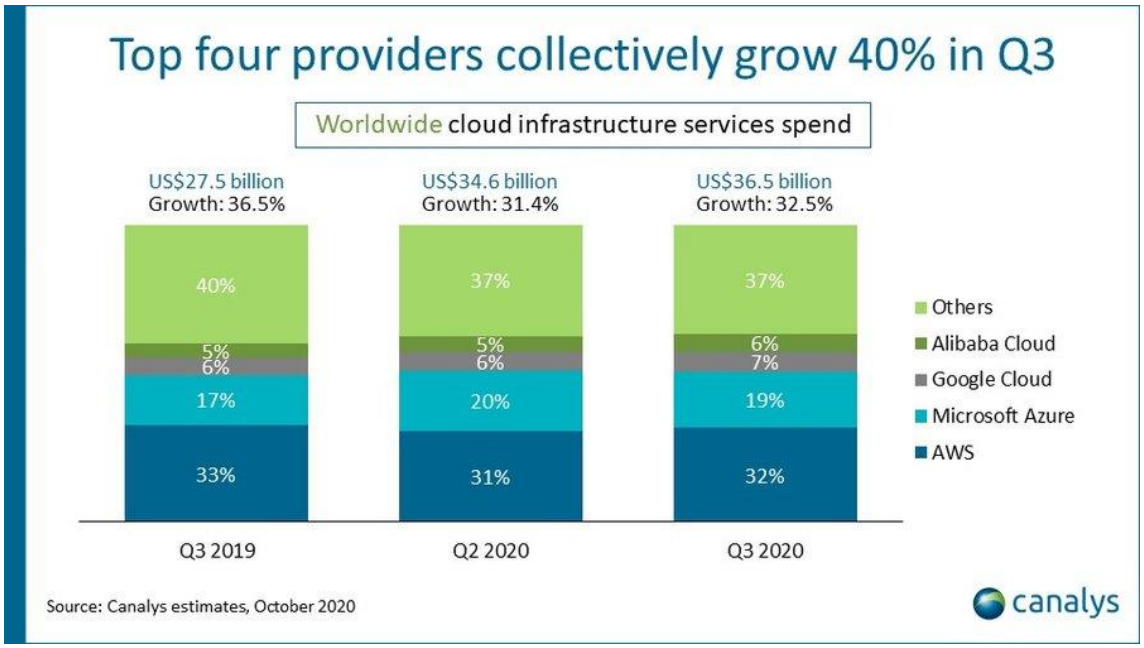
Amazon Web Services, también conocido por su acrónimo, AWS. Es el segundo proveedor de entornos *cloud* más importante del mundo solo por detrás de Microsoft, según Forbes (2017). No obstante, es el proveedor *cloud* del con mayor posicionamiento actual en el mercado, ya que, como puede verse en la **Figura 2**. AWS es el proveedor que más crecimiento experimenta año tras año, según el Cuadrante Mágico de Gartner (2020) para servicios de infraestructura y plataformas en la nube, dejando atrás a otros grandes competidores como Microsoft Azure o Google Cloud Platform.

Figura 2. *Cuadrante Mágico de Gartner que refleja el crecimiento de los proveedores cloud.* Gartner (2020)

Estos tres proveedores ocupan los primeros puestos en la esquina superior derecha del cuadrante de Líderes, otorgados por la Capacidad de Ejecución y la Integridad de la visión, asegurándose AWS el primer puesto por décimo año consecutivo.

Por otro lado, según el último estudio de Canalys y Synergy Research Group (2020), Microsoft Azure y AWS controlan de forma conjunta más del **50 %** del gasto mundial en servicios es infraestructura en la nube, como puede verse en la **Figura 3**. Y parece ser que esta tendencia va a continuar, esto se debe a que ambos proveedores continuamente están reforzando su posicionamiento en el mercado de los servicios *cloud*, gracias a sus grandes inversiones y la innovación que ofrecen a sus clientes.

Figura 3. *Porcentaje del gasto mundial en infraestructura cloud.* Canalys (2020)



Así mismo hay varios motivos por los cuales se ha elegido AWS frente al resto de sus competidores más próximos, como Microsoft Azure o Google Cloud Platform, las principales diferencias entre ambos se pueden consultar en la **Tabla 1**, los aspectos que se han tenido en cuenta a la hora de la comparación van desde los estratégicos o económicos, funciones de red, computación, almacenamiento de objetos, cifrado de datos, normas de cumplimiento o coste de los servicios (en cuanto a computación) que ofrecen.

Tabla 1. *Comparación de las características de los proveedores cloud AWS, Microsoft Azure y GCP.* Elaboración Propia

Características	**Amazon Web Services**	**Microsoft Azure**	**Google Cloud Platform**
Posicionamiento en el mercado	Primer puesto	Segundo puesto	Tercer puesto
Gasto mundial en servicios de infra.	32%	19%	7%
Cantidad de servicios ofrecidos	Primer puesto	Segundo puesto	Tercer puesto
Red cloud	Más de 230 puntos de presencia	Más de 170 puntos de presencia	Más de 144 puntos de presencia

Menor latencia	Primer puesto	Segundo puesto	Tercer puesto
Rendimiento máquinas IPS (1 núcleo)	Segundo puesto	Tercer puesto	Primer puesto
Rendimiento máquinas IPS (16 núcleo)	Primer puesto	Tercer puesto	Segundo puesto
Almacenamiento de objetos	5 tipos	5 tipos	5 tipos
Cifrado de datos	Tránsito y reposo AES256	Tránsito y reposo AES256	Tránsito y reposo AES256
Programas de conformidad	75 normas	91 normas	75 normas
Coste más bajo (computación)	Primer puesto	Segundo puesto	Tercer puesto

Viendo las comparaciones entre estos tres proveedores, Microsoft Azure y AWS son muy similares en cuanto a prestaciones de seguridad y almacenamiento. Es probable que Microsoft Azure sea una mejor opción de cara a sus programas de conformidad y marco legal (Microsoft, 2021), sin embargo AWS ha resultado ganador en cuanto a puntos de presencia en el mundo y a su menor latencia en el uso de los servicios, siendo este además, el que más servicios ofrece a un menor coste. Solo por debajo en cuanto a rendimiento de máquinas de 1 núcleo respecto a IPS (Instrucciones Por Segundo) debido a que Google Cloud Platform presenta un rendimiento de un 10% superior al de AWS (Cockroach Labs, 2021)

No obstante, si hubiera que elegir entre uno, AWS se erige como claro ganador, y por tanto ha sido elegido el proveedor *cloud* del sistema implementado.

Otras de las ventajas por las cuales se ha elegido AWS por su plataforma **flexible** diseñada para el uso de grandes, pequeñas y medianas empresas e incluso autónomos, ya que cuenta con la utilización créditos para y da la posibilidad de escalar los servicios en función de la demanda, de las cargas de trabajo o de la cantidad de recursos económicos que estén dispuestos a afrontarse.

Este proveedor de *cloud* sustenta de sus servicios a cientos de miles de empresas en **más de 245 países**, ofreciendo infraestructura de alta fiabilidad y fácilmente escalable, la razón de esto se debe a que funciona en 80 zonas de disponibilidad dentro de 25 regiones geográficas del mundo, como se puede ver en la **Figura 4**.

Figura 4. *Alcance de la red de AWS en el mundo.* Amazon Web Services (2021)

Al tener alcance en numerosas regiones en el mundo, esto le brinda la posibilidad de replicar los recursos en las zonas de disponibilidad que las contienen por lo que es sencillo **escalar** y reducir los recursos y ofrecer siempre una **disponibilidad** completa del sistema.

Por otro lado cuenta con una **seguridad** adaptada específicamente para los entornos *cloud*, supervisando y garantizando de forma ininterrumpida la integridad, confidencialidad y disponibilidad de sus servicios. Todos los datos que recorren sus servicios se encuentran cifrados tanto cuando se encuentran en reposo como cuando se encuentran en tránsito. Adicionalmente cuenta con la posibilidad de utilizar diferentes niveles de firewall a nivel de las instancias de computación, mediante el uso de los grupos de seguridad como a nivel de red con las listas de control de acceso a la red.

La infraestructura está diseñada para **rendimiento**. Esto es debido a que las regiones ofrecen baja latencia, baja pérdida de paquetes y alta calidad general de la red, gracias a su gran capacidad de red de fibra de 100 GbE totalmente redundante.

AWS es una plataforma que cuenta con varios sistemas de soporte y de autoayuda, como pueden ser su documentación o foros de debates. También es posible contratar planes de

soporte para contar con la ayuda de un experto y asesoramiento para la elección del plan que mejor se adapte a las necesidades del sistema que se desea implementar.

Finalmente, AWS da la posibilidad de elegir la región donde se quieren localizar los servicios de cara a la ubicación de los usuarios finales que van a acceder a estos, para así brindar de un mayor rendimiento y latencias de pocos milisegundos. O porque algunos de los servicios más recientes solo están disponibles en determinadas zonas hasta que se ha comprobado que están perfectamente testados.

3.3. Internet of Things

El "Internet de las cosas", también conocido por sus siglas en inglés IoT es un concepto ampliamente conocido hoy día. Según Gracia (2021) este término hace referencia a la agrupación e interconexión de dispositivos y objetos de una red donde todos ellos son visibles y pueden interaccionar entre sí. Por tanto el objetivo es una interacción máquina a máquina, sin necesidad de la intervención humana, esto se conoce como interacción M2M (*Machine to Machine*). La tecnología asociada al IoT permite la recogida de datos para posteriormente enviarlos a un sistema para su análisis o analizarlos para posteriormente enviarlos.

En este proceso de comunicación es donde IoT está evolucionando, ya que unos de los problemas que surgen es el protocolo de comunicación entre dispositivos, esto se debe a que la comunicación entre estos puede ser fácil y directa, sin embargo, otros pueden contar con protocolos no estándar y cuya conexión no es trivial.

Uno de los mecanismos que se ha intentado establecer es la creación de un protocolo abierto y estándar, propuesto por IBM y conocido como MQTT (*Message Queue Telemetry Transport*), este protocolo permite que todos los fabricantes puedan fabricar dispositivos que lo soporten, permitiendo así la comunicación entre dispositivos de diferentes fabricantes.

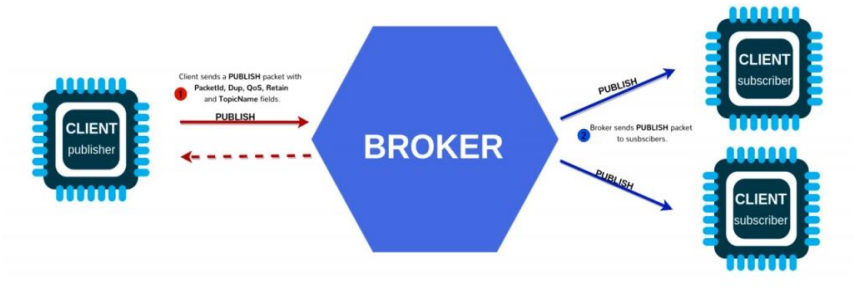
Por otro lado, los dispositivos IoT deben consideran diversos aspectos, como el bajo consumo y que sean de pequeño tamaño. Otra parte importante se encuentra en los sensores, el procesador y la plataforma encargada de gestionar la información. Y finalmente, otra parte importante se encuentra en la tecnología utilizada para la comunicación entre los dispositivos, donde se pueden encontrar las redes WiFi (consumo alto y bajo alcance) o redes móviles como 4G o 5G que cuentan con un mayor alcance y un menor consumo.

El protocolo MQTT ha sido elegido el protocolo de comunicación entre los dispositivos hardware por varios motivos, entre los que se encuentran los siguientes:

- Es un protocolo **ligero y sencillo**, siendo muy frecuente su utilización en dispositivos de baja potencia, como es el caso de este sistema. La escasa potencia se traduce en un menor consumo de energía, por lo que las baterías son más duraderas.
- Permite el envío de mensajes a través de la red, utilizando un **ancho de banda mínimo**, lo que facilita el envío al receptor en redes inalámbricas o con problemas de calidad.
- Dispone de medidas de medidas adicionales para **mejorar su calidad**, como protocolos para asegurar la calidad de los envíos (QoS) o su seguridad (SSL/TLS, autenticación, etc.).
- Al ser uno de los protocolos M2M más conocidos, cuenta con una gran comunidad que participa activamente en su **evolución y mejora** y, además, la documentación es extensa. Todo esto origina un protocolo suficientemente testado y consolidado para aportar robustez y fiabilidad a cualquier sistema IoT.

El funcionamiento de este protocolo se basa en un servicio de mensajería *push*, con patrón publicador/suscriptor (*pub-sub*), donde en este tipo de arquitectura los clientes se conectan a un servidor central denominado *broker*, por lo que la topología utilizada se trata de una en estrella, como puede verse en la **Figura 5**.

Figura 5. *Topología protocolo MQTT.* HWlibre (2020)



Los clientes inician una conexión TCP/IP con el *broker*, quien mantiene un registro de los clientes conectados, estos envían los mensajes en *topics*, por lo que otros clientes pueden suscribirse a ese *topic* y al bróker le llegarán los mensajes de suscripción. La estructura de estos *topics* cuenta con una estructura jerárquica.

Esta conexión se mantiene abierta hasta que el cliente la finaliza. La implementación utilizada de este protocolo es la versión segura, a través del puerto 8883 y funcionando sobre TLS, es decir, MQTTS.

Finalmente, cabe destacar, que la utilización de esta tecnología se decidió de forma intencionada tras compararlo con otras opciones como Zigbee, IFTTT o Bluetooth, donde resultó el más apropiado, como puede verse la **Tabla 2**.

Tabla 2. *Comparación de diferentes tecnologías de IoT.* Elaboración Propia

Tecnología	Hardware	Energía	Puntos fuertes
Zigbee	Necesidad de un HUB que controle la red de dispositivos	Consumo alto de energía	Muy implementado hoy día en dispositivos inteligentes.
Bluetooth	Dispositivo de radio y controlador digital	Consumo muy alto de energía	Muy extendido y establecido en el mercado.
IFTTT	No necesarios extras	Consumo alto de energía	Incipiente en la domótica y con soporte multiplataforma.
MQTT	No necesarios extras	Consumo bajo de energía	Sencillo y ligero, con medidas adicionales (calidad y seguridad) gran comunidad.

4. Técnicas y Herramientas

En este apartado se va a describir las principales técnicas y herramientas utilizadas para la implementación del sistema, incluyendo los lenguajes de programación, el entorno de desarrollo y las ventajas que este ofrece, la modularidad del proyecto, y herramientas utilizadas.

4.1. Aplicación web y *frameworks* de desarrollo

El sistema implementado se compone de una aplicación realizada con PHP, lenguaje de código abierto y débilmente tipado especialmente destinado para el desarrollo web, debido a su alta integración con HTML y JavaScript.

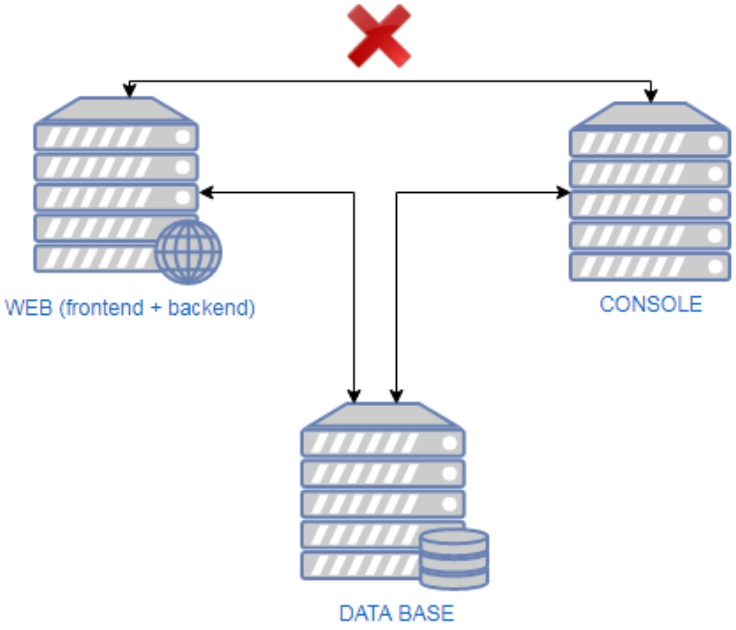
Una vez se teniendo claro el lenguaje de programación, es necesario establecer un entorno de trabajo donde ponerlo en práctica, por lo que se decidió utilizar un *framework* de desarrollo de software para mejorar la eficiencia en la creación del sistema, y aportar una mejor calidad, fiabilidad y robustez a este.

Hoy en día, existen muchos *frameworks* para desarrollo de PHP, algunos de estos son: Laravel, Symphony, Cake, Phalcon, Slim… Todos estos aportan diferentes características en el desarrollo y mejoran la eficiencia del programados, sin embargo, para el desarrollo del sistema se ha decidido utilizar el *framework* de Yii, concretamente la versión dos.

Yii2 permite tener varias aplicaciones web en el mismo proyecto, lo cual permite el desarrollo sencillo de una parte para acceso de clientes (*frontend*) y una parte privada para gestión de este (*backend*), adicionalmente cuenta con una aplicación para el sistema (*console*) para realizar tareas periódicas mediante el *crontab* o labores de mantenimiento y gestión de la parte web.

Dividir el sistema en varias aplicaciones lo dota de mayor modularidad, capacidad de mantenimiento y seguridad, como puede verse en la **Figura 6**. Todo lo anterior se ve potenciado gracias a una gran activa comunidad que provee de extensiones y *plugins* que potencian la funcionalidad del sistema y agilizan el desarrollo de nuevas implementaciones.

Figura 6. *Modularidad de aplicación en Yii2 y relaciones existentes.* Elaboración Propia



Con las opciones de caché activadas, Yii puede soportar nueve veces más peticiones por segundo que sus competidores, aunque, sin tener en cuenta el caché, sigue ofreciendo mejores prestaciones (Soltel, 2011). Yii es un framework rápido gracias a que las librerías no se cargan hasta que son utilizadas, por este motivo ha sido dotado de grandes prestaciones en distintos benchmarks realizados (PHP Benchmarks, 2017).

Este framework también permite la generación de código de la capa de acceso a base de datos, al igual que permite generar los CRUD (*Create Read Update Delete*) de todas las tablas del sistema, habiendo posteriormente, que adaptar los requisitos.

Por otro lado, al igual que con PHP se utiliza el Yii2, Bootstrap ha sido elegido como el framework que potencia y agiliza el desarrollo con HTML y CSS. Este provee estilos y funcionalidades ya construidas y preparadas para ser utilizadas sin apenas esfuerzo, por otro lado, Bootstrap hace uso de JavaScript nativo para sus funcionalidades, y conseguir de esta manera ser más abierto y tener menos conflictos con las herramientas que lo incorporan, como sí ocurre con otros *frameworks*.

Finalmente, para el desarrollo en JavaScript se ha utilizado Jquery, el cual presenta como grandes características la completa integración con el lenguaje AJAX, utilizado para la recuperación y envío de datos entre el servidor y el cliente de forma totalmente transparente para este. Adicionalmente, Jquery cuenta con extensiones y *plugins* que permiten mejorar las funcionalidades y suplir carencias del *framework* base.

4.2. Herramientas utilizadas

En este apartado se recogen las herramientas software (CASE, de seguimiento, de desarrollo, de pruebas …) y lenguajes de programación utilizados a lo largo del desarrollo del sistema.

REM

Herramienta experimental gratuita de Gestión de Requisitos diseñada para soportar la fase de Ingeniería de Requisitos de un proyecto. La versión de esta herramienta utilizada es la 1.2.2 y su propósito es la construcción de tablas de los requisitos del sistema. Se ha decidido utilizar esta y no otra debido a su sencillez en la interfaz y que es una herramienta con la que se está familiarizado.

EzEstimate

Esta herramienta permite realizar la estimación del esfuerzo de un proyecto a partir de los puntos de casos de uso de este. De nuevo el motivo de su utilización frente a otras es que ha sido utilizada otras veces y sus prestaciones son las adecuadas, la versión utilizada es la Lite 1.1.2.

Tom's Planner

Es una herramienta de gestión online y de uso gratuito que permite realizar la planificación temporal a partir de la creación de Diagramas de Gantt. Al ser una herramienta gratuita, sencilla y con las funcionalidades que se necesitaban se ha decidido utilizar su uso frente a otras similares que eran necesarias licencias.

Visual Paradigm

Herramienta utilizada a través de la licencia educativa que ofrece la Universidad de Salamanca basada en el lenguaje UML y utilizada para la creación de diferentes diagramas: secuencia, despliegue, casos de uso, de clase, etc. Ha sido utilizada, en la versión 15.0, debido a la confianza existente con la aplicación.

GitLab

Esta herramienta web de planificación y seguimiento de tareas se utiliza para dotar una visión más gráfica en el desarrollo del sistema. Haciendo uso, entre otros del método Kanban. Y también utilizada como servicio web de control de versiones, permitiendo separar el proyecto en dos ramas (*branches*) permitiendo trabajar con el proyecto en fase estable (*master branch*) y en fase de desarrollo (*develop branch*) a la vez y sin que existan conflictos entre estas. Principalmente se ha elegido su uso frente a otras plataformas por la sencillez que presenta, familiaridad y que es una herramienta gratuita.

TortoiseSVN

Se trata de un cliente SVN (subversión) gratuito que permite un control de versiones del sistema y la subida de este versionado al repositorio de GitLab empleado para el almacenamiento versionado de este. Es una aplicación gratuita y sencilla, con la que existe confianza en su utilización, la versión empleada es la 2.10.0.2.

Visual Studio Code

Es el editor de código utilizado en la implementación del sistema, es de código abierto y gratuito y permite una programación avanzada. Una de las mejores características de este editor es IntelliSense, que permite resaltar la sintaxis del código fuente para que aparezca en él, además permite el uso de funciones de autocompletar, basándose en variables, definiciones y módulos. Otra de sus principales características es la gran cantidad de lenguajes de programación con los que puede trabajar. La versión utilizada es la 1.56.2.

Hosts File Editor

Esta herramienta permite la edición del archivo *hosts* de Windows para así conseguir la resolución de nombres de dominios los cuales no están publicados en Internet. Como el objetivo era la realización de pruebas sin tener que publicar el dominio, esta simple herramienta cumple su función. La versión utilizada es la 1.2, siendo una versión gratuita.

MobaXterm

Esta herramienta gratuita, en la versión 20.2, ha sido utilizada para el control los servidores debido a la gran cantidad de herramientas que aporta, tanto para el control de estos como para la monitorización de los recursos. Permite el uso de determinadas

funciones de Linux, como el uso de comandos para controlar el sistema operativo desde teclado y permite conexiones remotas por terminal, escritorio remoto, conexiones remotas por FTP, SFTP y también permite el uso de nuevas funciones y herramientas mediante el uso de plugins

Arduino IDE

Es el IDE, con versión 1.8.9, utilizado para el desarrollo del programa y la carga de este en la placa NodeMCU utilizada como prototipo de dispositivo de envío de mensajes. Al ser una herramienta sencilla, gratuita y familiar en su utilización unido a que cuenta con grandes prestaciones, han sido las razones de su utilización.

Consola de administración de AWS

Esta aplicación engloba y hace referencia a un amplio conjunto de consolas de servicios para la administración de Amazon Web Services. La página de inicio proporciona acceso a la consola de cada servicio, además de una interfaz de usuario intuitiva para explorar AWS y obtener consejos útiles.

Navegadores Web

Para las pruebas y posterior comprobación se han utilizado diferentes navegadores como Google Chrome, Safari o Firefox y comprobar el aspecto que tiene en estos el sistema web implementado.

Terraform

Herramienta que permite el despliegue de infraestructura a través de archivos en diferentes proveedores de entornos *cloud*. Se ha decido su utilización frente a otras como Cloud Platform de AWS porque puede ser utilizada en la mayoría de los proveedores *cloud*, además de ser una herramienta de código abierto con una gran comunidad que da soporte. La versión utilizada es la 0.7.

Ansible

Es un motor de automatización de TI que permite de forma automatizada el aprovisionamiento de las infraestructuras desplegadas en la nube. Se ha decidido su utilización frente a otras porque es una herramienta gratuita que se ha utilizado otras

veces, cuenta con roles para agrupar las funcionalidades y cuenta con una gran comunidad contribuyendo a su soporte. La versión utilizada es la 2.7.

Vagrant

Esta herramienta gratuita permite la generación de entornos de desarrollo virtuales a partir de configuración en ficheros, utilizada en el entorno demo del sistema. Su uso se debe a la facilidad que brinda la utilización de entornos virtualizados, ya que es más rápido que carga una imagen en VirtualBox o VMWare y permite una sencilla y rápida configuración gracias a sus ficheros VagrantFile. La versión utilizada es la 2.1.4. Esta herramienta es muy útil en entornos de desarrollo montado por varias personas, ya que se asegura que todos ellos trabajen con el mismo entorno además de solucionar problemas de compatibilidad y de versionados

Adobe XD

Herramienta utilizada para el diseño de las interfaces utilizadas en el sitio web y la generación de las imágenes utilizadas como los logos del sistema. Permite crear imágenes preliminares de la interfaz de algún proyecto y permite ahorrar tiempo, ya que da la posibilidad de realizar cambios rápidos en la imagen preliminar en caso de ser necesarios, antes de llegar a la etapa de programación

Fritzing

Este programa de código libre ha sido utilizado, en la versión 0.9.6, para la realización de los diseños electrónicos del prototipo hardware implementado para el envío de mensajes. Es una herramienta sencilla, gratuita y de uso intuitivo que ha sido utilizada con anterioridad, por lo que se ha elegido para su utilización frente a otras.

KeePass

Es un gestor de contraseñas, utilizada en la versión 2.45, que permite el almacenamiento de estas de forma segura, utilizado para almacenar las contraseñas de los servicios utilizados como credenciales de AWS, bases de datos, servicios externos, etc. Debido a su sencillez, que es una herramienta gratuita y familiaridad en la utilización, se ha elegido para almacenar las credenciales.

Qualys SSL Labs

Esta herramienta online y gratuita permite comprobar la seguridad del servidor web y dotarlo de una nota en función de la calidad de la seguridad de este, teniendo en cuenta aspectos como: Certificado, protocolos de *handshaking*, *cipher suites*, etc. Debido a su gran utilización por una gran comunidad y soporte se ha decidido su utilización para comprobar la seguridad del sistema.

Google Meet

Herramienta online y gratuita utilizada para videoconferencias online y poder así mantener reuniones con los tutores de este proyecto.

Git

Esta herramienta gratuita de versionado de código permite mantener el sistema en un servidor, y controlar los cambios y actualización que se realizan en él, permitiendo así volver a un punto anterior en el desarrollo o el trabajo en paralelo, para posteriormente fusionar diferentes versiones en un mismo punto.

PHP

Lenguaje de desarrollo web utilizado en la implementación del sistema, lenguaje de código abierto, que cuenta con un gran número de librerías y extensiones, lo que hace que aumente sus funcionalidades y prestaciones. Cuenta con el soporte de una gran comunidad, la cual participa y mejora esta tecnología de forma activa. Alguna de sus ventajas son que se trata de un lenguaje multiplataforma y muy débilmente tipado, lo que dota de una mayor flexibilidad en la implementación de funciones.

HTML y CSS

Lenguajes utilizados para la construcción de la página web y dotación de las propiedades y características como, el posicionamiento en la página, tipo de letra, tamaño y fuente, colores, etc. El uso de estos lenguajes permite mostrar la información con un formato responsivo al dispositivo que lo presenta y de forma *friendly* con el usuario.

JavaScript

Este lenguaje de programación está orientado a entornos web y permite a estos que exista dinamismo e interacción con el usuario.

Frameworks Web (Bootstrap 3 y Jquery)

Estos dos *frameworks* son los utilizados para el desarrollo del código HTML + CSS y JavaScript respectivamente. Cuya explicación se encuentra en el apartado 4.1 de esta memoria.

AJAX

Lenguaje utilizado para la recuperación y envío de datos desde y hacia el servidor de forma transparente para el usuario y de forma autónoma.

SQL

Lenguaje de gestión de BDD utilizado en el servidor de bases de datos MySQL implementado, se trata de un lenguaje relacional basado en tablas.

YML

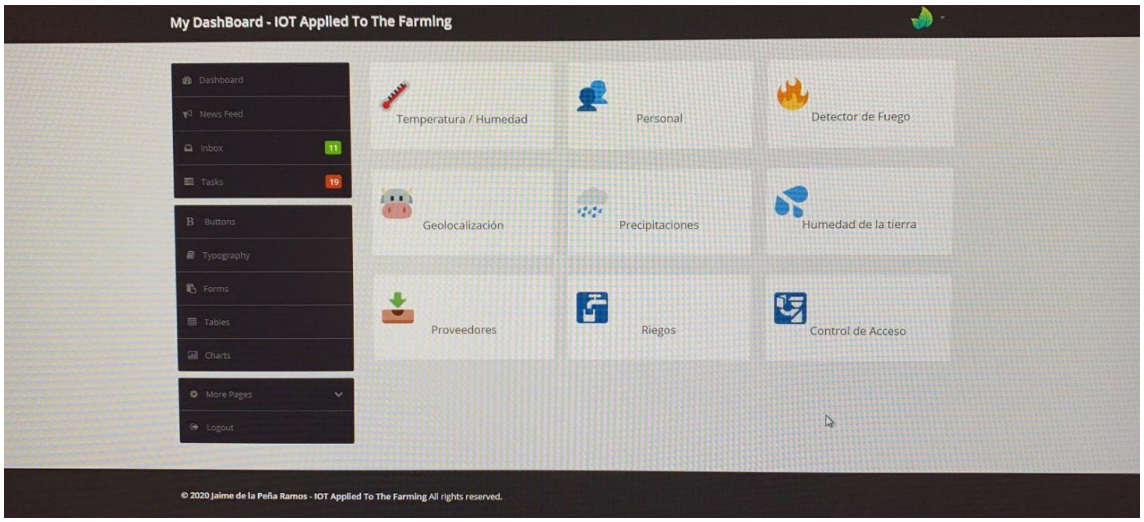
Es el lenguaje utilizado para la implementación en de las tareas Ansible, su sintaxis es encilla y fue diseñado para que fuera muy legible. Utiliza una notación basada en sangría.

5. Aspectos Relevantes

5.1. Seguimiento del sistema

El desarrollo de este proyecto comenzó en junio de 2020 con una implementación basada en PHP nativo, sin utilizar ningún *framework* de desarrollo ni aplicación de patrones arquitectónicos y exclusivamente con el uso de algún servicio de AWS (como servidor web, base de datos y almacenamiento de ficheros). Tampoco contaba con las funcionalidades de control y gestión de usuarios, que sí presenta ahora. Esta versión se representa en la **Figura 7**.

Figura 7. *Primera versión del sistema con PHP sin framework de desarrollo.* Elaboración Propia



Tras una reunión mantenida con los tutores, se llegó al acuerdo de utilizar un *framework* de desarrollo, con un patrón arquitectónico establecido y almacenando los datos enviados en una base de datos, ya que anteriormente, los datos se encontraban en ficheros exclusivamente.

Por lo que daba comienzo la elaboración de un nuevo sistema con la planificación temporal que se podrá consultar en el **Anexo I** que complementa la memoria.

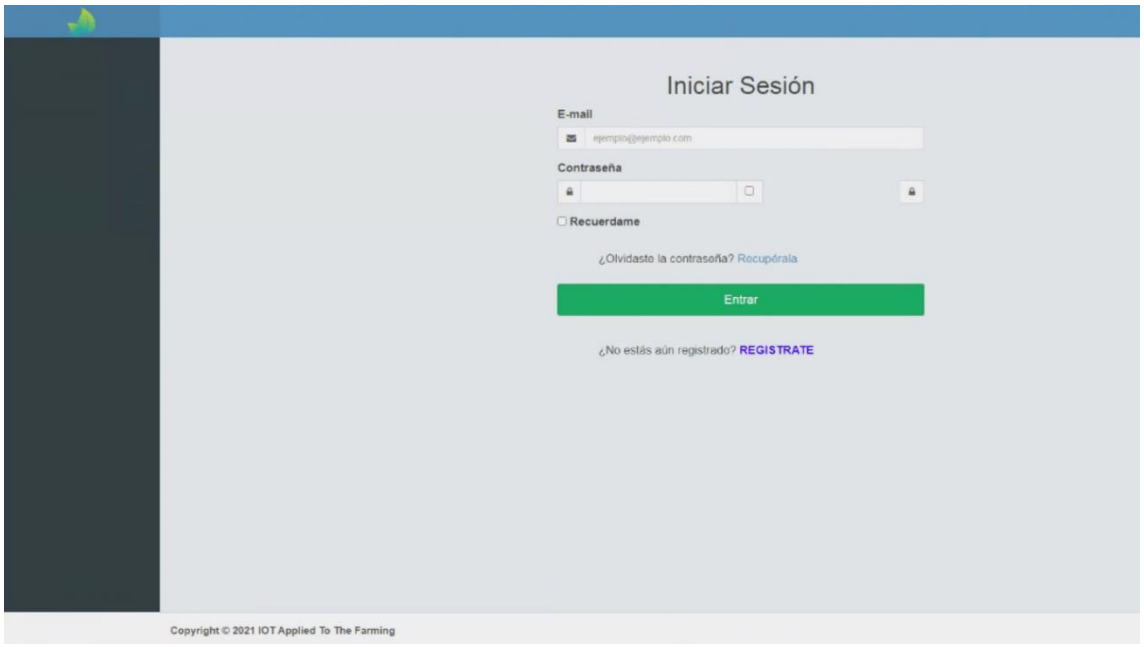
Ya que había que realizar un nuevo enfoque del sistema, se comenzó con la captura de los requisitos de este, los cuales se encontrarán detallados en el **Anexo II** de la memoria. Añadiendo una gestión y control de usuarios y demás funcionalidades como la representación gráfica o exportación de datos cuya implementación se vio favorecida por la utilización de un framework. De esta forma, surge una nueva estimación del esfuerzo, que se podrá encontrar de forma más detallada en el **Anexo I** de la memoria.

Una vez se tenían claros los casos de uso y los objetivos del sistema, se podía hacer la agrupación en paquetes que los aúnen y separen en función de sus características, esta información puede ser complementada con el **Anexo II** de la memoria. Por lo que se tenían claros los modelos a realizar y las interfaces y controladores a implementar.

Por otro lado, el diseño del sistema se encuentra basado en un patrón arquitectónico principal que sería el Modelo-Vista-Controlador, acompañado de otros patrones como el *Abstract Factory* o el DAO, de nuevo, se podrá más información en el **Anexo III** que complementa la memoria.

De esta forma nace la segunda versión del sistema, la cual se encuentra más estructurada y cuya representación se encuentra en la **Figura 8**.

Figura 8. *Tercera versión del sistema con framework de desarrollo y dashboard.* Elaboración Propia

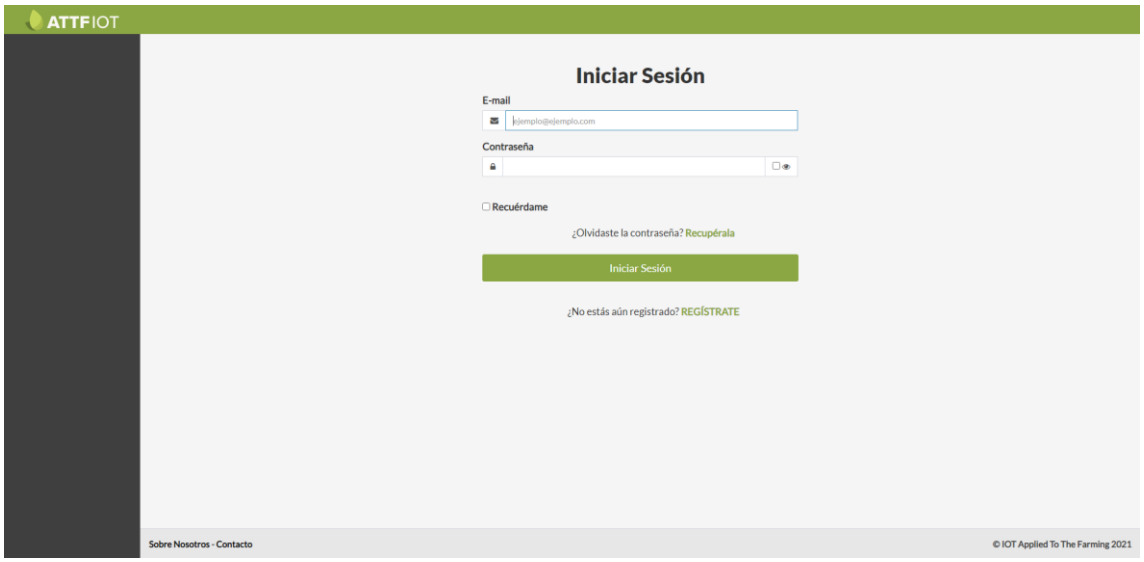


Esto también requería el uso nuevas tecnologías, por lo que se decidió utilizar más servicios de los cuales ofrece AWS, como puede ser su servicio de caché basado en Redis, para el almacenamiento de la sesión entre otras cosas, o una forma de compartir el código entre los servidores, haciendo uso de su sistema de ficheros compartido, o el envío de correos a través de servicio autogestionado. Todos estos servicios y más se podrán encontrar detallados en el **Anexo III** que complementa la memoria.

Ahora que ya se contaba con una estructura y funcionalidades definidas, había que comenzar con la interacción del usuario, para la cual se han implementado unas guías

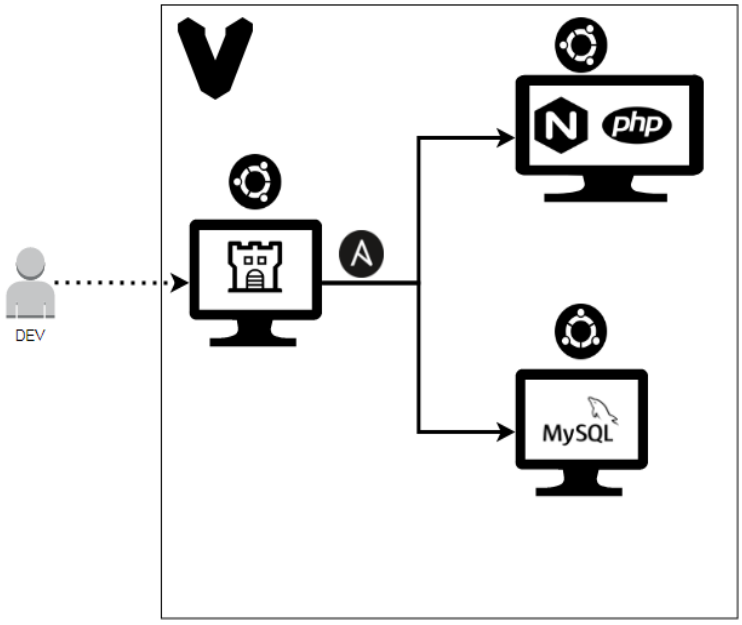
básicas de UX acompañadas de una adecuada UI. Toda esta información se podrá encontrar de forma más completa en el **Anexo III** de la memoria. Por lo que la representación de la interfaz final del sistema, con los colores que mejor lo representan, este se encuentra en la **Figura 9**.

Figura 9. *Versión final del sistema*. Elaboración Propia



El proceso de desarrollo de la aplicación web ha sido desarrollado en un entorno demo, el cual ha sido construido a partir de la herramienta Vagrant, instalada en el equipo de desarrollo, y la cual permite generar entornos virtuales a través de imágenes ya configuradas y descargadas a partir de un repositorio conocido como Vagrant Cloud, estas imágenes cuentan con el sistema operativo Ubuntu 18.04, con una configuración básica preestablecida, por lo que era necesario provisionar este entorno demo utilizando la herramienta de Ansible, para así contar con una instancia de gestión, otra que actuará como servidor web y una última que se utilizará para el almacenamiento del sistema de gestión de bases de datos. Si así se desea, se puede obtener más información sobre el funcionamiento del entorno demo dirigiéndose al **Anexo III** de la memoria. El diagrama de arquitectura de este sistema se encuentra en la **Figura 10**.

Figura 10. *Diagrama de arquitectura entorno demo.* Elaboración Propia



Tras, completar la implementación de la aplicación web, se decidió que se pudiera acceder a esta a través de un nombre de dominio, por lo que se adquirió el dominio "iotappliedtothefarming.com" en IONOS para que se pudiera acceder al sistema a través de Internet sin necesidad de realizar configuraciones extras.

Por otro lado, se ha implementado un prototipo, el cual será explicado en el apartado 5.4 de esta memoria, el resto de los dispositivos hardware se podrán encontrar de forma más detallada en el **Anexo III** que acompaña a esta memoria.

Finalmente, hay que añadir que se añaden dos anexos más, el **Anexo IV** que recoge una guía para el programador y el **Anexo V** donde se podrá encontrar un manual para el usuario.

5.2. Metodología

En este apartado, se va a exponer una concisa perspectiva de la metodología de trabajo empleada en el sistema, esta será expandida en su totalidad en los anexos de que complementan la memoria.

5.2.1. Metodología de trabajo

La metodología escogida para el desarrollo del sistema ha sido el Proceso Unificado Ágil (PUA), esta es una versión más actualizada y moderna del Proceso Unificado clásico. Y está más enfocada a los desarrollos ágiles, que son los más comunes en la actualidad.

Acompañando al PUA, se ha utilizado la metodología Kanban para la representación visual de las tareas.

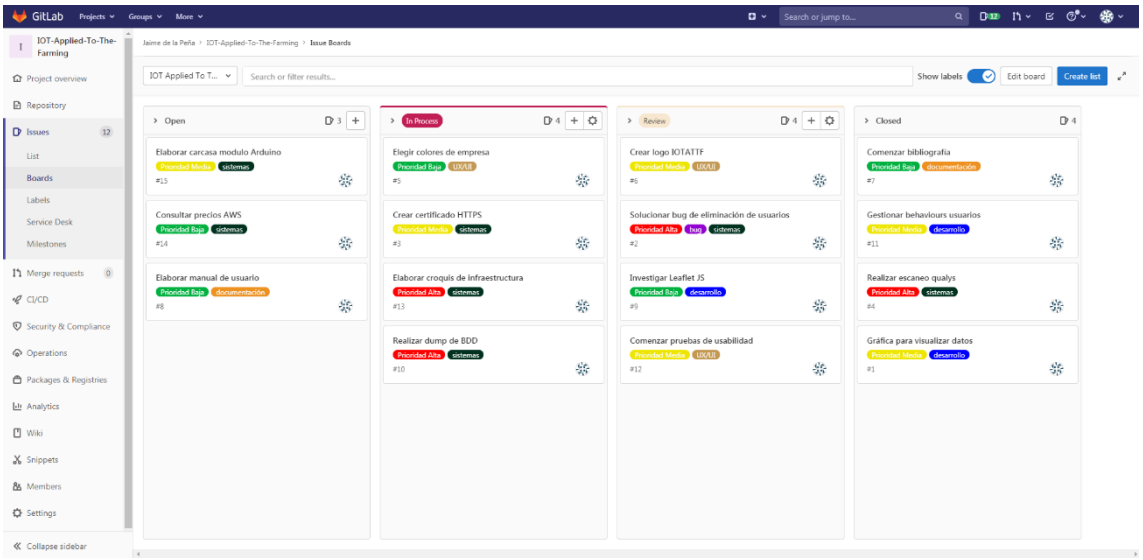
El motivo de su elección radica en que la elaboración de este sistema cuenta con una alta carga de investigación y documentación, donde ha sido necesario comprender el funcionamiento de un entorno *cloud*, el desarrollo de código a partir de un *framework* o la elaboración de un sistema IoT a partir del protocolo MQTTS. Por lo que no existen muchos requisitos que se puedan asegurar al principio del desarrollo, porque la mayoría de estos surgen a medida que se va avanzando en la elaboración del sistema y los cambios y adaptaciones serán, en su mayoría, frecuentes.

Iteraciones + Tareas y Sprints

A pesar de ser PUA una versión más moderna del Proceso Unificado clásico, sigue estando compuesto por cuatro fases bien diferenciadas: Inicio, Elaboración, Construcción y Transición, estas fases a su vez divididas en iteraciones que van incrementando con el producto final, y serán divididas en *sprints* de forma que en cada una de ellas se irán añadiendo tareas nuevas para satisfacer las necesidades que vayan surgiendo.

Para la representación de las tareas de cada *sprint* se ha utilizado la metodología Kanban, que consiste en una tabla dividida en filas y columnas, donde las columnas representan las fases y las filas las tareas. En la **Figura 11** se muestra la tabla empleada.

Figura 11. *Tablero utilizado en el método Kanban para la elaboración del sistema.* Elaboración Propia



Por otro lado, en la **Tabla 3** se encuentran los *sprints* resultante en la realización del sistema.

Tabla 3. *Sprints seguidos en la realización del sistema.* Elaboración Propia

ID	Nombre	Tiempo Estimado (días)	Tiempo Real (días)	Estado
1	**Sprint 1.- Inicio del proyecto**	8	10	
1.1	Investigación sistemas IoT similares	2	3	Realizada
1.2	Investigación proveedores cloud	3	4	Realizada
1.3	Investigar lenguajes de desarrollo web	2	2	Realizada
1.4.	Propuesta y diagrama de solución	1	1	Realizada
1.5	Entregable: Propuesta de solución			Entregada
2	**Sprint 2.- Aprendizaje y conocimiento tecnologías**	18	24	
2.1	Aprendizaje y documentación de PHP	5	7	Realizada
2.2	Aprendizaje y documentación AWS	10	15	Realizada
2.3	Aprendizaje y documentación protocolo MQTT	3	2	Realizada
2.4	Entregable: Propuesta de solución con AWS + PHP y IoT con MQTT			Entregada
3	**Sprint 3.- Desarrollo del sistema y pruebas**	59	53	
3.1	Implementación funcionalidades login, logout, register, reset, recover pass	12	10	Realizada
3.2	Pruebas ID 3.1	2	1	Realizada
3.3	Implementación funcionalidades lectura y gestión datos IoT	20	22	Realizada
3.4	Pruebas ID 3.3	3	2	Realizada
3.5	Implementación funcionalidades gestión usuarios	7	5	Realizada
3.6	Pruebas ID 3.5	1	1	Realizada
3.7	Construcción infraestructura AWS	5	3	Realizada
3.8	Despliegue del sistema	1	1	Realizada
3.9	Implementación resto de funcionalidades	5	3	Realizada
3.10	Pruebas de funcionamiento de todo el sistema	2	3	Realizada

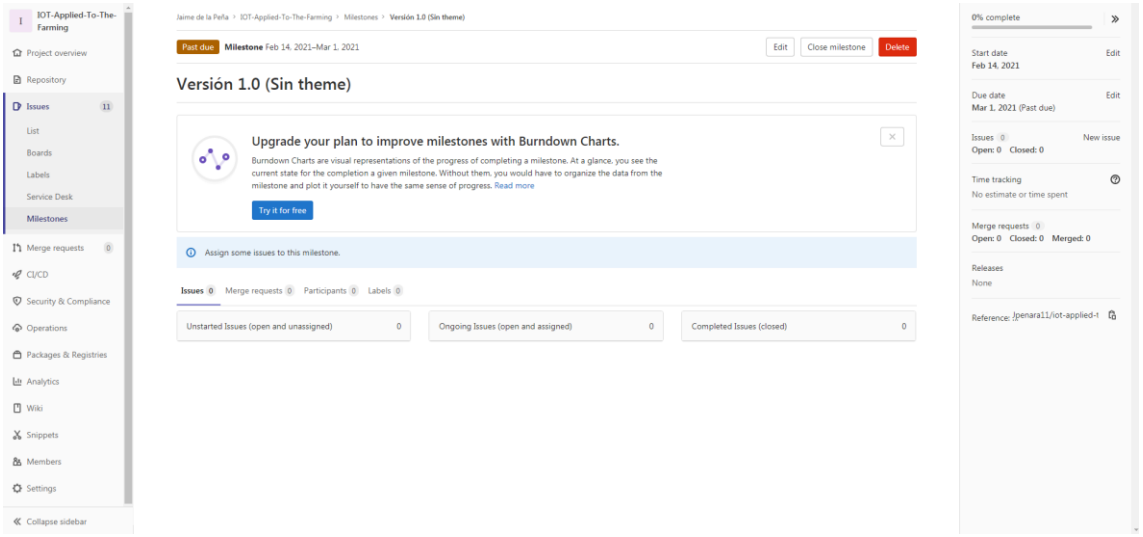
3.11	Pruebas ID 3.10	2	3	Realizada
3.12	Entregable: Primera versión del sistema			Entregada
4	**Sprint 4.- Adaptación del sistema a framework + nuevos servicios de AWS**	72	75	
4.1	Investigación *frameworks* de desarrollo web	3	2	Realizada
4.2	Aprendizaje y familiarización Yii2	10	7	Realizada
4.3	Funcionalidades CRUD de usuario	12	15	Realizada
4.4	Pruebas ID 4.3	3	5	Realizada
4.5	Funcionalidades CRUD de los datos	10	10	Realizada
4.6	Pruebas ID 4.3	5	4	Realizada
4.7	Funcionalidades gestión y control de usuarios	7	11	Realizada
4.8	Pruebas ID 4.7	4	4	Realizada
4.9	Adición de nuevos servicios AWS	8	5	Realizada
4.10	Pruebas y mejoras del sistema	10	12	Realizada
4.11	Entregable: Segunda versión del sistema			Entregada
5.0	**Sprint 5.- Mejora experiencia usuario del sistema y adaptaciones UI**	16	16	
5.1	Elección colores y fuentes del sistema	2	3	Realizada
5.2	Elección logos del sistema y realización	1	2	Realizada
5.3	Aplicación y adaptación de tema para uso de *dashboard*	4	4	Realizada
5.4	Aplicación de colores corporativos, logos, fuentes	1	1	Realizada
5.5	Elaboración análisis heurístico	1	1	Realizada
5.6	Mejoras del sistema tras el análisis heurístico	7	5	Realizada
5.7	Entregable: Tercera versión del sistema			Entregada
6	**Sprint 6.- Integración y despliegue**	23	28	
6.1	Despliegue del sistema en AWS	1	1	Realizada
6.2	Integraciones y adaptaciones del sistema en AWS	3	5	Realizada
6.3	Adaptaciones de los recursos de AWS	5	7	Realizada
6.4	Pruebas de las funcionalidades del sistema	4	4	Realizada
6.5	Corrección de errores y mejoras	10	11	Realizada

6.6	Entregable: Cuarta versión del sistema			Entregada
7	**Sprint 7.- Adquisición dominio del sistema y securización del entorno**	5	5	
7.1	Adquisición dominio del sistema y configuración DNS	1	1	Realizada
7.2	Análisis seguridad del servidor	2	2	Realizada
7.3	Mejora de la seguridad del servidor	2	2	Realizada
7.4	Entregable: Quinta versión del sistema			Entregada

Iteraciones + Hitos

Como ya se ha explicado anteriormente, este proceso se repite a lo largo de los diferentes ciclos de desarrollo de los cuales se compone el sistema, donde cada uno de estos ciclos está formado por las diferentes fases explicadas mencionadas, por último, cada una de estas fases siguen iteraciones que finalizan con un hito cada una, como puede verse en la **Figura 12**.

Figura 12. *Hito de la primera versión del sistema sin tema.* Elaboración Propia

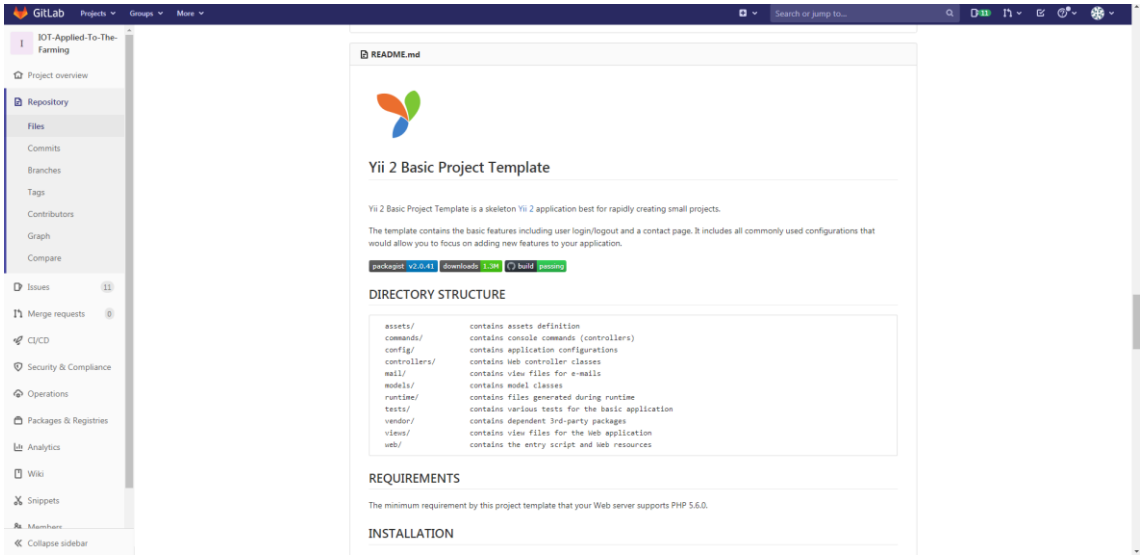


Indicando así, el comienzo de la siguiente iteración y controlando el desarrollo del sistema. Por otro lado, este proceso iterativo en el cual se van pasando por las diferentes fases de desarrollo y se van desplegando funcionalidades en el sistema, se ha realizado a través de una herramienta de subversión, TortoiseSVN, donde se subía el código a GitLab para, posteriormente, ser desplegado en el repositorio *bare* del servidor y de esta forma tener un control de versiones.

Desarrollo dirigido por pruebas + Refactorización de la BDD

Como ya se ha explicado anteriormente, es posible que a lo largo del desarrollo del sistema surjan nuevos requisitos los cuales no estaban planificados en un principio. Para estos casos se ha utilizado la técnica de desarrollo dirigido por pruebas o *Test-Driven Development* (TTD) de la metodología PUA. Esta consiste en escribir las pruebas de la funcionalidad que se quiere implementar de forma unitaria, en el entorno de pruebas, se asegura que la prueba falla (si no falla es porque el requisito estaba implementado o la prueba ha sido errónea) para posteriormente refactorizarla y que pase las pruebas de forma satisfactoria. Esto ha sido posible debido a que se utilizaba dos entornos, con diferentes ramas, como se puede ver en la **Figura 13**, un entorno de desarrollo o pruebas donde se testean estas nuevas funcionalidades y un entorno final o productivo, donde el código que se aloja en este se encuentra estable y probado correctamente.

Figura 13. *Código desplegado en el repositorio de GitLab y estructura en ramas.* Elaboración Propia



Por otro lado, para la implementación de estas nuevas funcionalidades es probable que sea necesario modificar la estructura de la base de datos, de hecho, esta metodología lo contempla. Para estos casos Yii2 soporta la característica llamada migración de base de datos, la cual permite tener un seguimiento de esos cambios, cuyo versionado es controlado junto al código fuente.

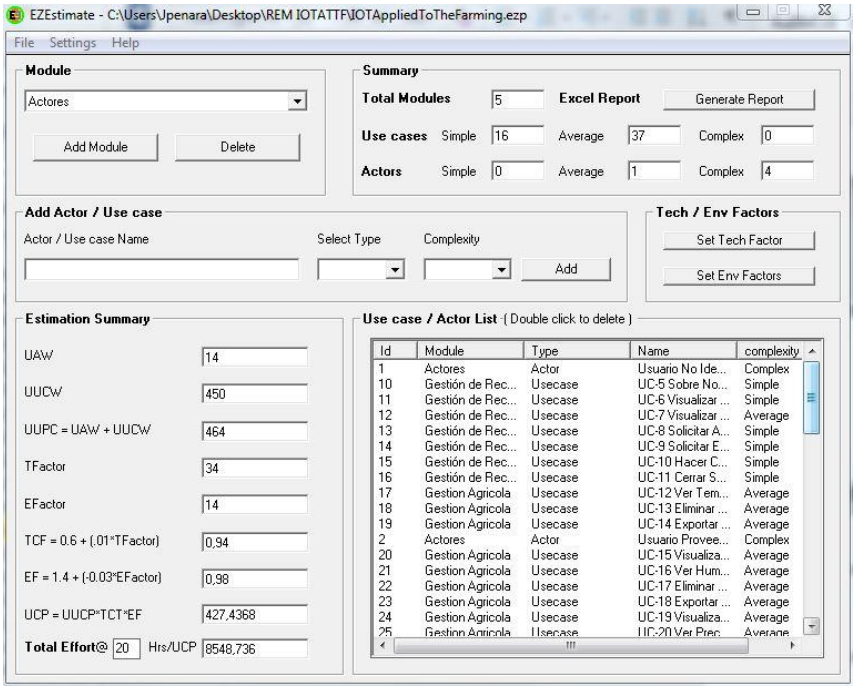
5.2.2. Estimación del esfuerzo y planificación temporal

A continuación, se va a introducir la estimación del esfuerzo y planificación temporal planteadas en el desarrollo del sistema, las cuales serán detalladas en su totalidad en el **Anexo I** de la memoria.

Para determinar el esfuerzo necesario para la elaboración del sistema que se plantea se utilizará una medida funcional del tamaño de este, denominada análisis de puntos de casos de uso (UCP), esta es calculada a través de estimaciones. Teniendo en cuenta factores como el número y complejidad de casos de uso, número de complejidad de los actores y otros factores técnicos y del entorno.

Tras las estimaciones realizadas se obtiene que las horas necesarias para elaborar el sistema (TPH), es de 8548,736 horas/UCP, es decir, 948 días. Este resultado ha sido obtenido, además de los métodos tradicionales, con la herramienta EZEstimate. Como se muestra en la **Figura 14**.

Figura 14. *Estimación del esfuerzo para el desarrollo del sistema en la herramienta EZEstimate.* Elaboración Propia



Una vez realizada la estimación del esfuerzo para el desarrollo del sistema, se realiza una planificación de las tareas que se deben realizar, para ello se ha utilizado la herramienta de gestión online, Tom's planner, el cual posee numerosas características para distribuir los recursos y organizar las tareas. La temporización está basada en jornadas completas, no obstante, es una estimación ya que las tareas pueden tener mayor o menor complejidad y por tanto no es posible realizar una estimación exacta. A continuación, se muestra una figura con las tareas planificadas y la distribución que se ha tenido en cuenta.

Figura 15. *Planificación temporal para el desarrollo del sistema en la herramienta Tom's Planner.* Elaboración Propia

Activity	Inicio	Final	Días
Inicio del proyecto, documentación y aprendizaje	**01-07-20**	**06-10-20**	**193.0**
Arduino (módulos, ESP8266, protocolos, manuales)	01-07-20	16-07-20	12.0
PHP (sintásix básica, funcionamiento, versiones)	01-07-20	30-07-20	22.0
Yii2 (frameworks, guía básica, uso de vendors)	22-07-20	03-09-20	32.0
Protocolo MQTT (que és, funcionamiento, arquitectura, uso)	06-07-20	23-07-20	14.0
HTML, CSS y JavaScript (uso de BootStrap. buenas prácticas)	03-07-20	05-08-20	24.0
AWS (guía básica, uso de consola, servicios y documentación)	01-07-20	06-10-20	70.0
RaspBerry Pi (conexiones, funcionamiento, SSOO)	07-07-20	31-07-20	19.0
Especificación de requisitos	**06-07-20**	**24-09-20**	**128.0**
Realizar BrainStrom inicial de funcionalidades y requisitos	06-07-20	22-07-20	13.0
Realizar documento de requisitos	08-07-20	26-08-20	36.0
Identificar los casos de uso del sistema, requisitos de información, no funcionales etc.	07-07-20	31-07-20	19.0
Revisar la documentación realizada	10-08-20	17-09-20	29.0
Adaptar y/o modificar la documentación realizada	13-08-20	24-09-20	31.0
Validar y aceptar los requisitos establecidos en la documentación			
Análisis	**18-08-20**	**19-11-20**	**118.0**
Definir el contexto y descripción detallada del mismo	18-08-20	23-09-20	27.0
Elaborar el diagrama de clases del sistema	01-09-20	22-09-20	16.0
Elaborar los diagramas de secuencia del sistema	07-09-20	15-10-20	29.0
Elaborar los diagramas de clases de análisis del sistema	17-09-20	19-11-20	46.0
Elaborar el documento final de análisis			
Diseño	**23-09-20**	**12-11-20**	**56.0**
Diseñar la base de datos del sistema	23-09-20	08-10-20	12.0
Extraer las clases de diseño, adaptar y conformar la base de datos del sistema	28-09-20	15-10-20	14.0
Diseñar la arquitectura del sistema	15-10-20	29-10-20	11.0
Diseñar la interfaz del sistema	19-10-20	12-11-20	19.0
Elaborar el documento final de diseño			
Implementación	**07-10-20**	**14-04-21**	**256.0**
Implementar dispositivo Hardware	21-10-20	11-11-20	16.0
Implementar servidor On-Cloud	03-11-20	10-03-21	92.0
Implementar protocolo MQTT con QoS 2	07-10-20	28-10-20	16.0
Implementar aplicación web	13-10-20	14-04-21	132.0
QA & Mantenimiento	**24-11-20**	**28-05-21**	**196.0**
Testeo y pruebas del dispositivo Hardware	24-11-20	25-12-20	24.0
Testeo y pruebas del servidor On-Cloud	02-02-21	26-05-21	82.0
Testeo y pruebas del protocolo MQTT	01-12-20	22-12-20	16.0
Testeo y pruebas de la aplicación web	16-02-21	28-05-21	74.0
			947.0

Como se puede apreciar en la **Figura 15**, muchas de las fases se solapan en el tiempo, esto es debido a que a pesar de que la fase de documentación es meramente formativa, se comienza a implementar funcionalidades o a realizar labores de configuración. Al tratarse de un desarrollo cíclico, con ensayo y error, la fase de implementación se extiende en la gran parte del sistema, y donde en cada fase específica, se encuentra una iteración de aseguramiento de la calidad para cerrarla.

Finalmente, en la **Figura 15**, se puede hacer una división en grupos de tareas para, así, conseguir una planificación más legible y que el reparto de horas y tareas sea más conciso, donde cada grupo se corresponde con una fase diferentes del proyecto, algunas solapadas, otras dependientes de las predecesoras, de forma que hasta que no eran finalizabas, no daba el comienzo de la siguiente fase.

5.2.3. Especificación de requisitos

En este apartado se plantea el proceso de captura de requisitos, a través del método propuesto por Durán y Bernárdez (2000), haciendo uso de diagramas de casos de uso para la ilustración de los requisitos funcionales.

El método de Durán y Bernárdez está basado en la utilización de un sistema con plantillas genéricas, las cuales ya se encuentran elaboradas para la correcta descripción de los requisitos funcionales, no funcionales, de información, objetivos, actores del sistema, etc.

Este proceso será descrito en detalle en el **Anexo II** de la memoria.

A continuación, en la **Tabla 4** representa la plantilla utilizada en este método para la captura de requisitos funcionales.

Tabla 4. *Ejemplo de plantilla, a partir del caso de uso UC-0001, para la recolección de requisitos a través del método de Durán y Bernárdez.* Elaboración Propia

UC-0001	**Iniciar Sesión**
Versión	1.0 (04/05/2021)
Autores	• Jaime de la Peña Ramos
Fuentes	• Jaime de la Peña Ramos
Dependencias	• [OBJ-0009] Gestión de Recursos Humanos • [UC-0002] Registrarse • [UC-0003] Recordar Contraseña
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el usuario pulsa sobre el botón "Iniciar Sesión" una vez ha rellenado correctamente los campos necesarios. o durante la realización de los siguientes casos de uso: [UC-0002] Registrarse, [UC-0003] Recordar Contraseña, [UC-0004] Contacto, [UC-0005] Sobre Nosotros.
Precondición	El usuario no ha iniciado sesión en el sistema.

Secuencia normal	Paso	Acción
	1	El sistema una vez ha validado los campos necesarios para el inicio de sesión, le permitirá acceder al usuario.
	2	Si el usuario selecciona el botón "Registrarse", se realiza el caso de uso Registrarse (UC-0002).
	3	Si el usuario selecciona el botón "Recordar Contraseña", se realiza el caso de uso Recordar Contraseña (UC-0003).
Postcondición	Dependerá de la elección del usuario.	
Excepciones	Paso	Acción
	1	Si las credenciales de acceso no son las correctas, el sistema se lo notificará al usuario por pantalla, a continuación este caso de uso continúa.
Rendimiento	Paso	Acción
	1	1 segundo(s)
Frecuencia	10 veces por minuto(s)	
Importancia	vital	
Urgencia	inmediatamente	
Estado	en construcción	
Estabilidad	media	
Comentarios	Ninguno	

5.2.4. Análisis del sistema

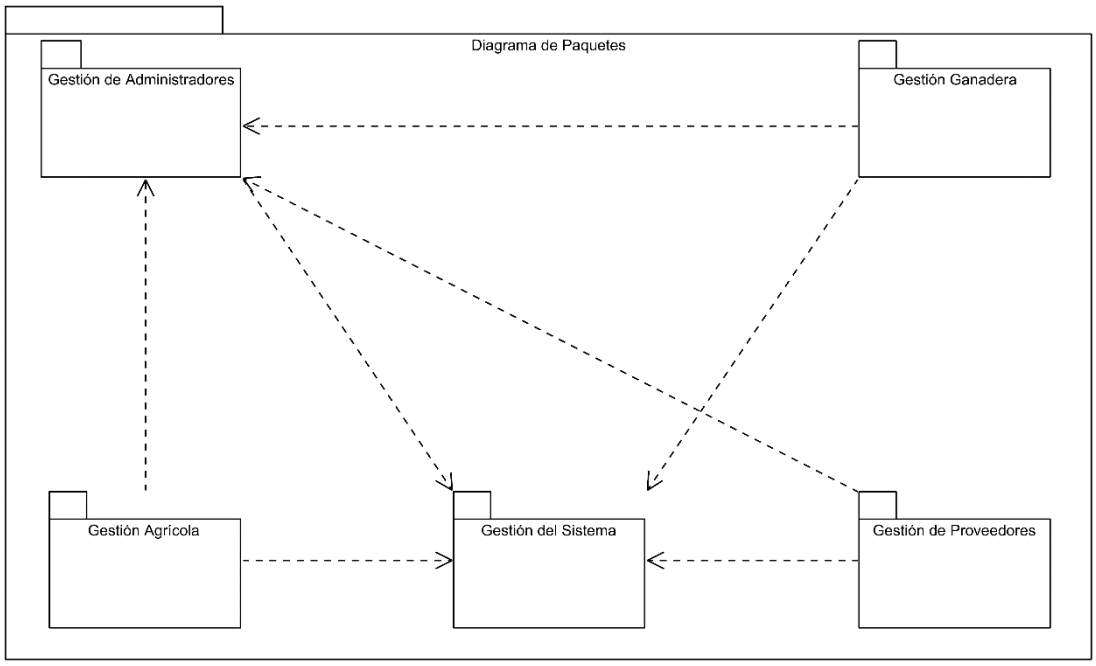
Esta fase consiste en la descomposición del sistema en paquetes, haciendo una distinción entre las clases conceptuales, o clases de análisis, que no tienen correspondencia con las clases reales del producto porque se encuentran en una capta de abstracción más arriba. Por lo que en esta fase se realiza un análisis de qué clases se van a representar en el sistema y como organizarlas en paquetes de forma que exista concordancia entre ellas, para más tarde, partir de esta base e ir bajando niveles de abstracción.

Por lo que esta fase solo está compuesta de interfaces, controladores y entidades. Donde a cada uno de los paquetes les corresponde un conjunto de casos de uso de los cuales tendrán que detallar su funcionalidad.

La información completa de esta fase de análisis se puede encontrar en el **Anexo II** de la memoria.

El sistema actual se descompone en paquetes bien diferenciados, la **Figura 16** muestra el diagrama de paquetes empleados en el sistema.

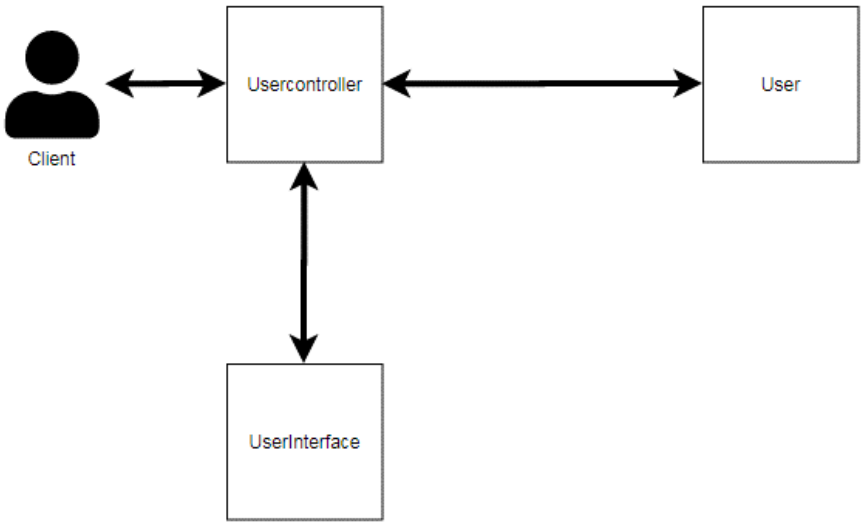
Figura 16. *Diagrama de paquetes del sistema.* Elaboración Propia



5.2.5. Diseño del sistema

Tras concluir la fase de análisis, da comienzo la de diseño. Donde toda la información de nuevo puede encontrarse de forma completa en el **Anexo III** de la memoria. En esta fase se define la arquitectura final del sistema, los patrones empleados y cómo se han implementado. En este caso, el patrón elegido es el **Modelo-Vista-Controlador** (MVC), el cual puede verse representado en la **Figura 17**, a través de una representación con la clase *User* implementada en el sistema, y en contiene la información del usuario registrado.

Figura 17. *Patrón MVC empleado en el sistema representado a través de la clase User*. Elaboración Propia

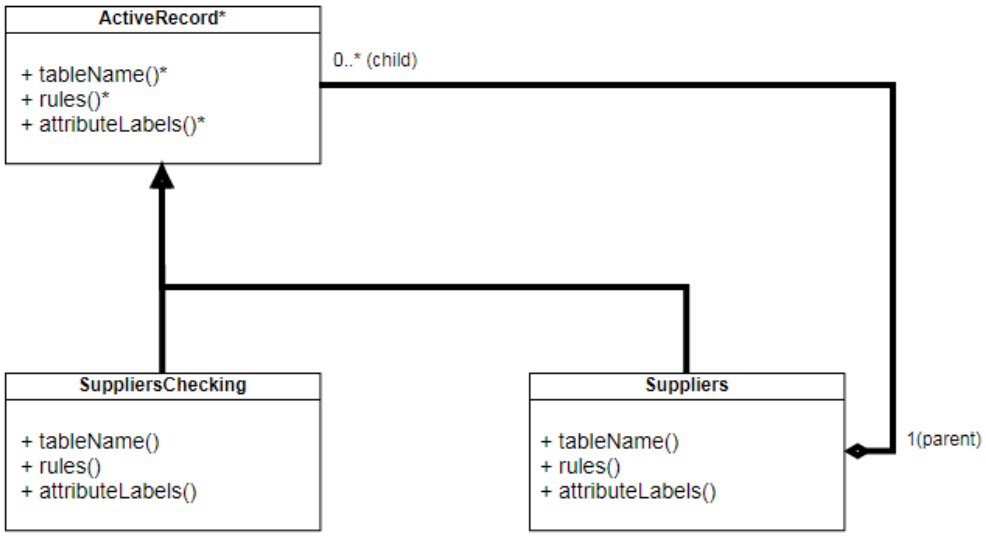


Dado que es el que más se adaptaba a las necesidades de la arquitectura y al estilo del *framework* de desarrollo utilizado (Yii2), esto se debe a que se trata de un sistema basado su funcionamiento en el acrónimo CRUD (*Create, Read, Update, Delete*) conformado por las funciones básicas que lo componen.

Complementando al patrón MVC, se han utilizado también los patrones **DAO** (*Data Acess Object*), para realizar la gestión de la base de datos, ***Abstract Factory*** y ***Composite***. Siendo estos dos últimos esenciales para el desarrollo con el *framework* antes mencionado, ya que ayudan en el uso de la herencia y extensión de funcionalidades en clases heredadas.

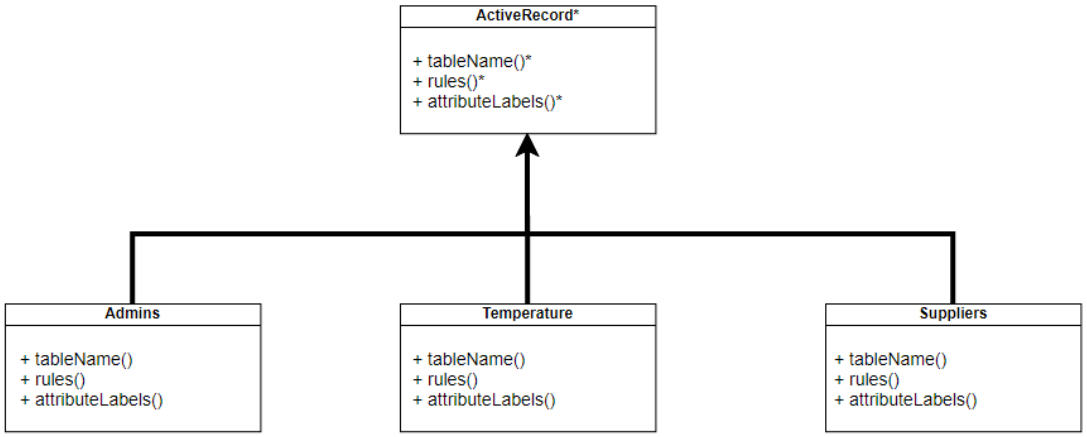
El patrón ***Composite*** permite la construcción de objetos complejos a partir de otros más simples y similares entre sí, gracias a su diseño y estructura en forma de árbol, como puede verse en la **Figura 18**.

Figura 18. *Patrón Composite empleado en el sistema.* Elaboración Propia



Por otro lado, el patrón ***Abstract Factory***, abstrae la estructura de clases para así favorecer la herencia, de esta manera, engloba en una clase superior los métodos que implementarán sus cases heredadas y así acceder a estos métodos de forma transparente, tal y como puede verse en la **Figura 19**.

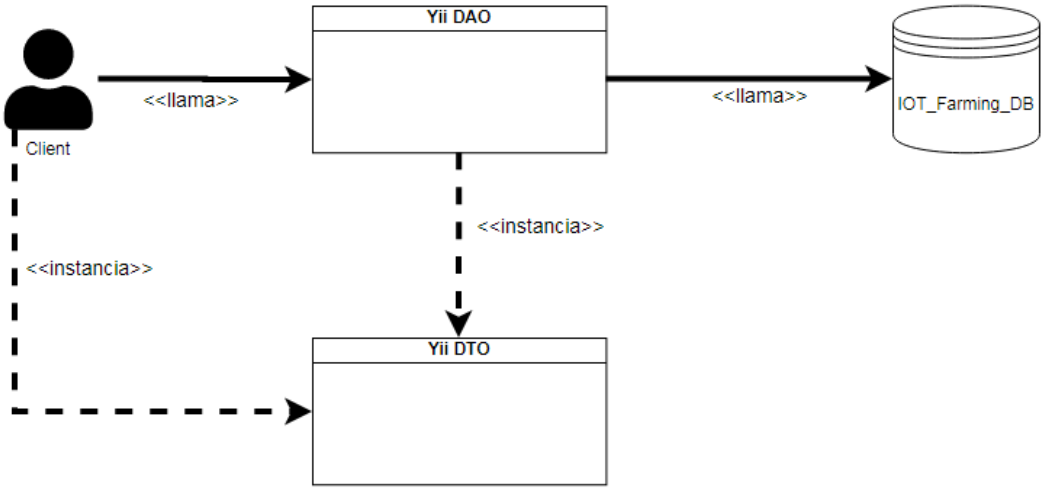
Figura 19. *Patrón Abstract Factory empleado en el sistema.* Elaboración Propia



Finalmente, el patrón **DAO**, está orientado a la gestión de bases de datos, ya se consiste en el acceso a una clase frontal, la cual cuenta con unos métodos que abstraen al usuario

de la lógica existente en la clase principal, es decir, el usuario hace uso de métodos de esta clase principal sin saber a qué tipo de base de datos se conecta ni como esta implementa la lógica de acceso. Este esquema se puede visualizar en la **Figura 20**.

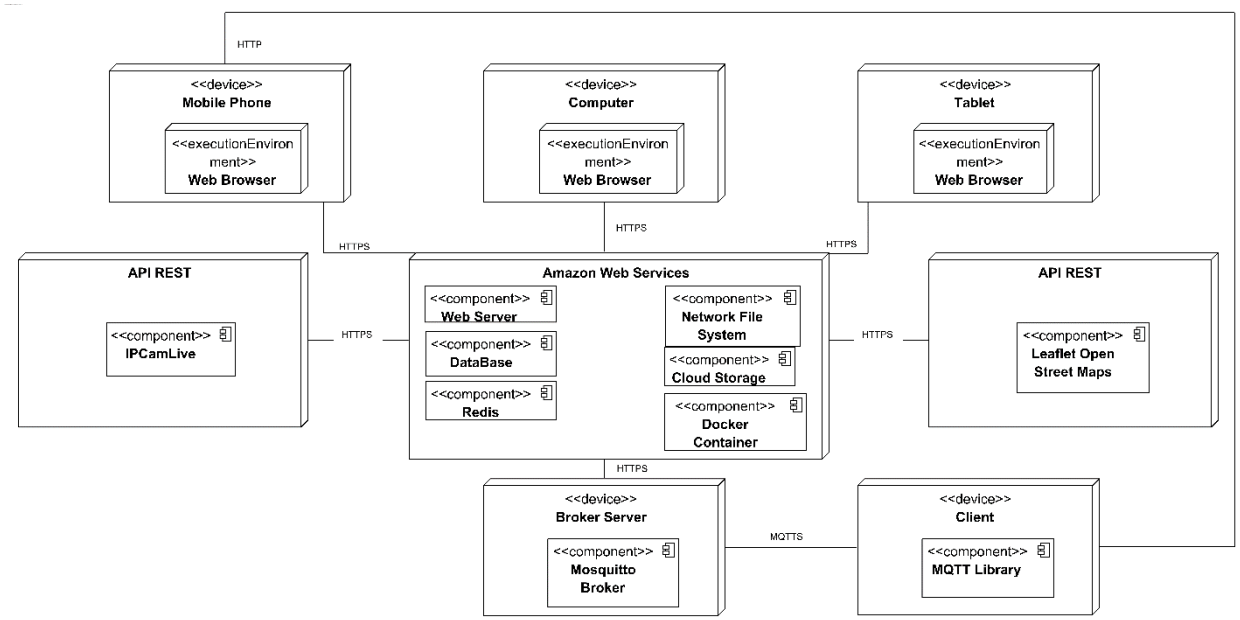
Figura 20. *Patrón DAO empleado en el sistema para acceder a la BDD.* Elaboración Propia



5.3. Componentes Software

A partir del diagrama de despliegue, representado en la **Figura 21**, se va a describir brevemente los elementos que componen el sistema, para más información se puede consultar el **Anexo III** que complementa la memoria.

Figura 21. *Diagrama de despliegue del sistema.* Elaboración Propia



La aplicación web implementada en Yii2 será desplegada en un servidor web, el cual está alojado en el proveedor *cloud* AWS, a través de una herramienta de control de versiones con Git. Una vez el código se encuentra desplegado, es necesario añadirlo al Network File System de AWS para que todas las instancias cuenten con la última versión del código.

Por otro lado, el envío de los datos monitorizados lo realizará un cliente (configurado previamente por el usuario), al servidor *broker* a través del protocolo MQTTS ya explicado anteriormente, este servidor se sincronizará con el almacenamiento de ficheros de AWS para que ambos tengan el mismo contenido.

La aplicación web accederá a las APIs de *Leaflet Open Street Maps* para la representación geográfica en un mapa de las coordenadas enviadas y, por otro lado, *IPCam Live* podrá mantener una visualización en tiempo real de una cámara IP, esta visualización se empotrará en un *iframe* implementado en la aplicación.

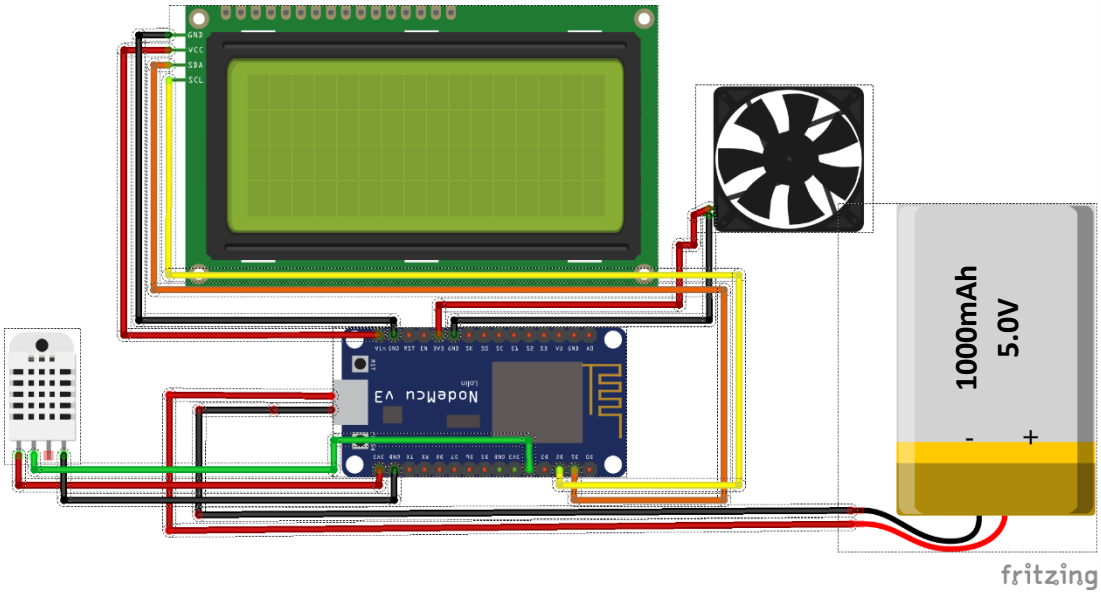
Por otro lado, existe la posibilidad de añadir funcionalidades adicionales a través de microservicios, los cuales son implementados mediante imágenes de Docker y desplegados en el contenedor de imágenes de AWS, conocido como ECS (*Elastic Container Service*). Un ejemplo de este, y que se encuentra desplegado, es un contenedor que contiene una imagen de un antivirus (ClamAV) que detecta la presencia de virus a través del escaneo de ficheros. Realizando una llamada API a este contenedor, con el fichero, devolverá una respuesta para saber si este es seguro, en caso de no serlo, no sería procesado. Este contenedor se encuentra desplegado a través del motor Fargate que permite la ejecución de aplicaciones sin la necesidad de la gestión y administración de servidores, de esta forma se proporcionan tareas controladas en un entorno aislado.

Finalmente, el acceso a la aplicación web se hará a través de navegadores web que podrán encontrarse en diferentes dispositivos como ordenadores, *smartphones* o *tablets*.

5.4. Componentes Hardware

A continuación, en este subapartado, se va a explicar el prototipo para envío de datos implementado, el esquema de conexiones se puede apreciar **Figura 22**.

Figura 22. *Circuitería del prototipo hardware desarrollado.* Elaboración Propia



Este prototipo tiene como dispositivo principal la placa NodeMcuV3, esta placa de desarrollo integra el módulo ESP8266, este chip permite la conexión con una red WiFi. Esta placa se encuentra insertada en una estructura que le brinda un mayor soporte, y una sujeción óptima, al tener una base más estable.

Adicionalmente a esta versión 3 de la placa, existen otras que se diferencian en cuanto a las conexiones de los pines y las prestaciones, por estos motivos se ha optado por utilizar esta versión.

La LCD, de tipo 20x4, se encuentra conectada con un adaptador I2C, con el fin de reducir el número de salidas de esta y simplificarlos a cuatro pines (alimentación, tierra, SDA y SCL) cuenta también con un potenciómetro que permite regular el brillo de la pantalla. El pin de alimentación de la LCD tiene que ir conectado al pin Vin de la placa NodeMCU ya que de esta forma se encuentra alimentado con un voltaje aproximado de 5V, ya que si se conecta a los pines clásicos de 3.3.V no será suficiente y por tanto no se verían correctamente la pantalla. Por otro lado, los pines SDA y SCL tienen que ir conectados a los pines D1 y D2 respectivamente, ya que corresponden con las salidas SDA y SCL de la placa. Además, para evitar un exceso de calor, y contribuir a la disipación de este se ha

añadido un ventilador alimentado con 3.3V y un disipador de aluminio al módulo ESP8266.

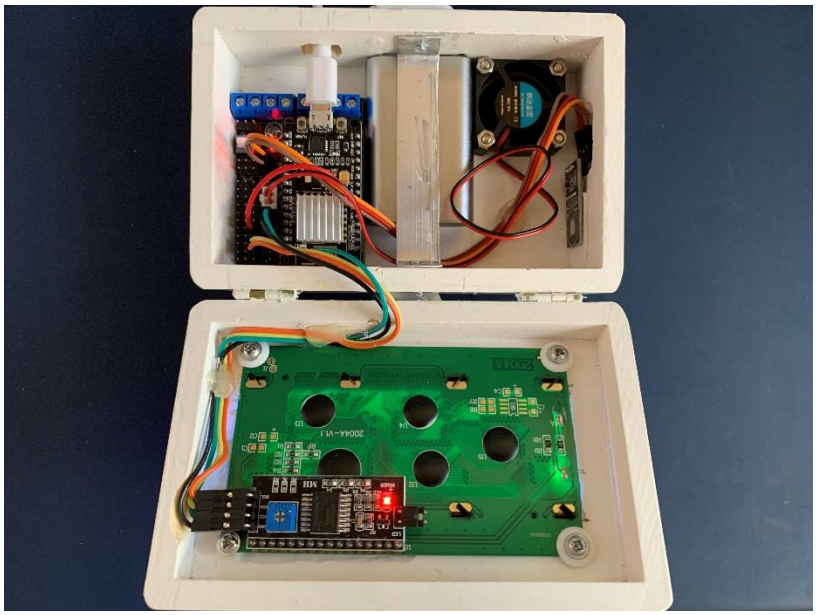
Por otro lado, el DHT 22 utilizado para medir la temperatura y la humedad, se conectará a los pines de 3.3V de la placa y tierra, la salida de este se podrá conectar a cualquiera de los pines digitales, en este caso se ha elegido que sea el D3.

Así mismo, la batería utilizada se trata de una *powerbank* conectada mediante cable de tipo microUSB a la placa, esta batería cuenta con un voltaje de 5V y una capacidad de 1000mAh, suficiente para alimentar toda esta circuitería.

Haciendo uso de materiales reciclados se ha diseñado una carcasa que contenga los elementos mencionados anteriormente y proteger así la circuitería y las conexiones entre los elementos, como se muestra en la **Figura 23** y **Figura 24**.

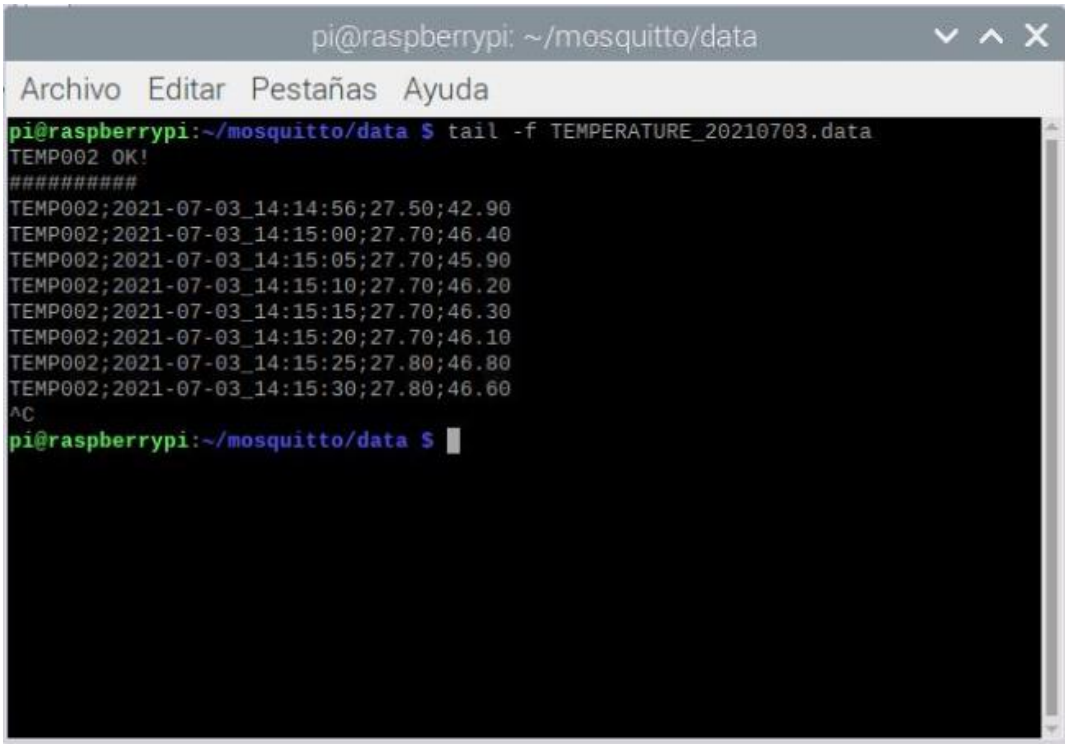
Figura 23. *Vista principal del prototipo implementado.* Elaboración Propia

Figura 24. *Vista interior de prototipo implementado.* Elaboración Propia



Finalmente, el servidor *broker* se encontrará alojado en una RaspBerry Pi 4. A través de un servicio que pone a este servidor en escucha, irá recibiendo los datos, estos se almacenarán en un fichero con nombre el tipo de dato y la fecha de recepción, tal y como se muestra en la **Figura 25**. La especificación técnica de la RaspBerry Pi 4 podrá ser consultada en el **Anexo III** de la memoria.

Figura 25. *Captura de pantalla de la RaspBerry Pi4 con los datos obtenidos del sensor*. Elaboración Propia



5.5. Despliegue en AWS

A continuación, en este subapartado, se va a describir el proceso de despliegue en AWS.

En primer lugar, es necesario explicar que se van a utilizar dos entornos y por tanto dos arquitecturas para el sistema. Estos son los siguientes.

- **Entorno pre-productivo (Staging Environment): Entorno para pruebas.**
- **Entorno productivo (Productive Environment): Entorno estable final.**

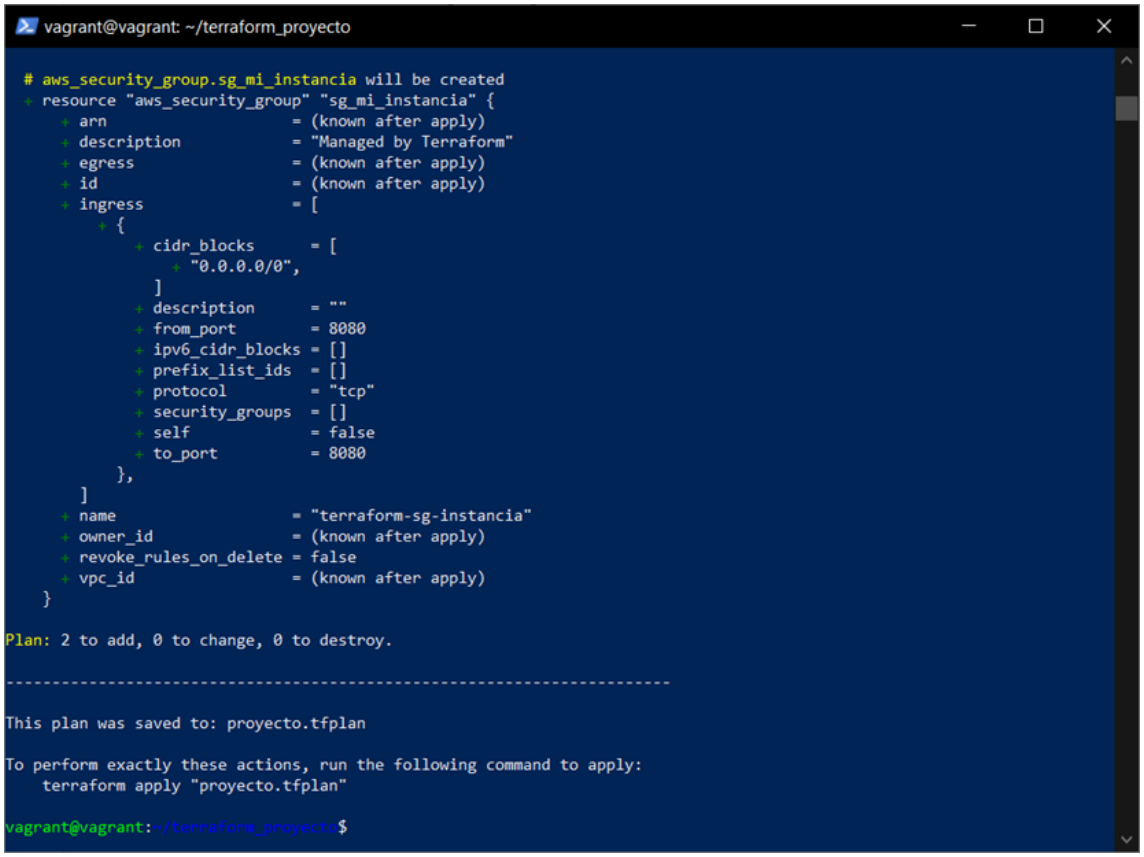
La diferencia entre estos dos entornos radica en que, en el primero de ellos, los recursos son más limitados y se pueden parar. Está pensado para probar las funcionalidades del sistema sin que esto incremente en los gastos. Por otro lado, los recursos utilizados son de niveles más bajos y, por tanto, prestaciones menores que en el entorno productivo.

Tanto el proceso de despliegue de la infraestructura en el entorno de pruebas como en el entorno productivo es realizado de forma automatizada a través de la herramienta Terraform. Esta herramienta de orquestación de código permite el despliegue de la infraestructura que se le indique, el cual, es similar al servicio *CloudFormation* de AWS con la ventaja de que esta herramienta se puede utilizar para el despliegue de infraestructura en cualquier proveedor *cloud*, por lo que de esta manera el proceso de despliegue no se encuentra acoplado al proveedor AWS.

Si bien, el despliegue con Terraform se hará a partir de ficheros, en los cuales se encuentran definidos los recursos a crear y las variables de configuración de estos. Al tener el código dividido en módulos reutilizables permite que puedan ser reutilizados haciendo pequeños cambios.

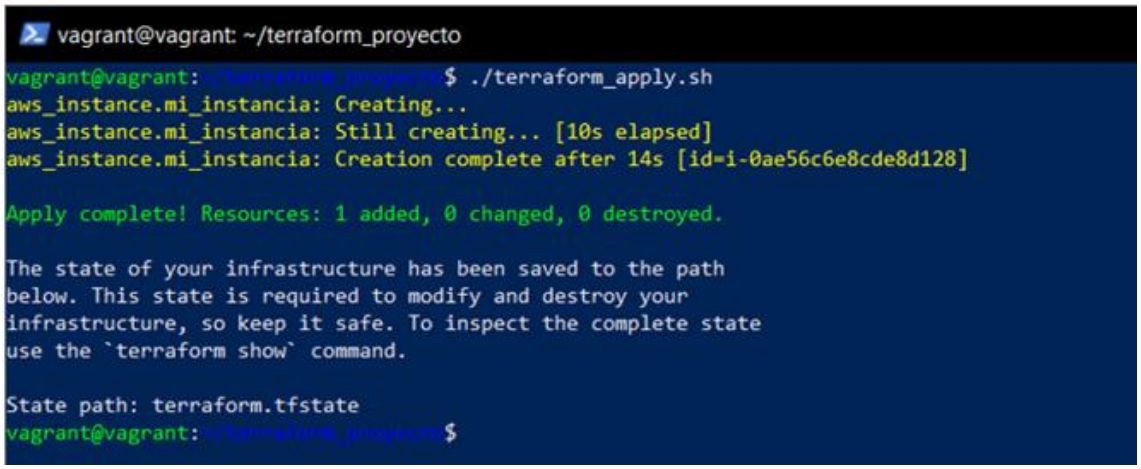
Una vez se tiene la configuración preparada, es necesario crear un plan de ejecución, el cual, previsualizará los recursos y la configuración a aplicar en estos, tal y como puede verse en la **Figura 26**.

Figura 26. *Ejecución comando de planificación de Terraform.* Elaboración Propia



Conforme con los recursos que se van a desplegar, hay que indicarle que los aplique, y, Terraform, se encargará de generar los recursos definidos en el fichero principal hasta llegar al estado objetivo, como se puede ver en la **Figura 27**. Una vez haya terminado, se habrá generado un fichero que almacena el estado actual de la infraestructura y este es consultado por Terraform para obtener la información existente antes de aplicar nuevos cambios sobre la misma, de esta forma se consigue no tener que desplegar todo de nuevo si se quieren añadir más recursos, ahorrando tiempo en el despliegue.

Figura 27. *Aplicación del comando que aplica el plan de ejecución en Terraform.* Elaboración Propia



Este proceso de despliegue de nuevo se podrá consultar de forma más detallada en el **Anexo III** de la memoria.

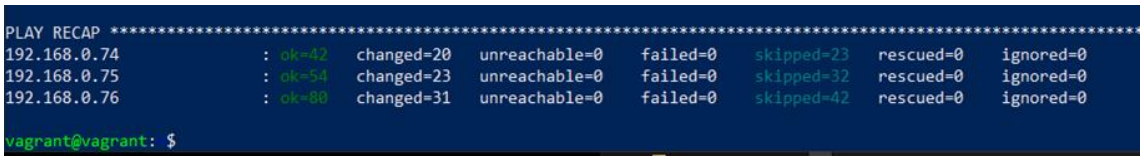
Una vez que se han desplegado los recursos en AWS, las instancias EC2 cuentan con la imagen base de Ubuntu 18.04, por lo que será necesario provisionarlas y configurarlas para su utilización. Este proceso se realiza a través de la herramienta Ansible, este software permite la gestión y configuración remota de servidores de forma automatizada. A través de la ejecución de un *playbook*, el cual es un fichero de configuración, implementación y orquestación, permite describir la configuración que se desea aplicar en los servidores gestionados. Este *playbook* cargará roles, tal y como puede verse en la **Figura 28**.

Figura 28. *Roles principales del playbook de ejecución del Ansible.* Elaboración Propia

```
1  ---
2  - hosts: localhost, nginx, mysql
3    become: yes
4    roles:
5      - dev-sec.os-hardening
6    vars:
7      sysctl_overwrite:
8        # Enable IPv4 traffic forwarding.
9        net.ipv4.ip_forward: 1
10
11 - hosts: nginx
12   become: yes
13   roles:
14     - geerlingguy.nginx
15
16 - hosts: mysql
17   become: yes
18   roles:
19     - geerlingguy.mysql
```

Estos roles contienen la información necesaria para: realizar la instalación de un servidor NGINX, la instalación de la tecnología PHP-FPM o la securización del servidor. Una vez se tiene la configuración preparada, hay que aplicarla para que sea ejecutada en todos los servidores que se especifica, tal y como puede verse en la **Figura 29**.

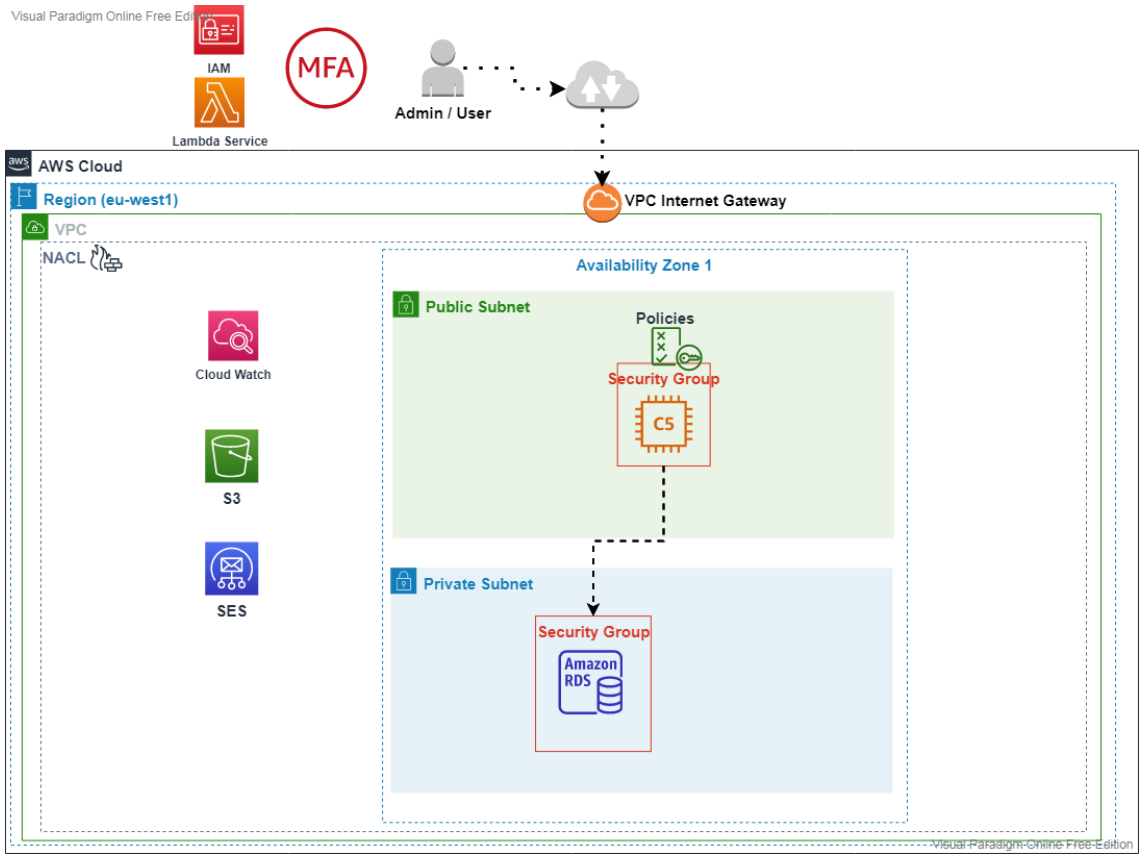
Figura 29. *Fin de la provisión del Ansible.* Elaboración Propia



Como se ha explicado al comienzo de este sub-apartado, se dispondrá de dos entornos para el despliegue del sistema.

El entorno pre-productivo está diseñado para ofrecer un funcionamiento básico del sistema. Una representación del diagrama de arquitectura de este entorno se puede visualizar en la **Figura 30**.

Figura 30. *Diagrama de arquitectura del sistema en el entorno pre-productivo.* Elaboración Propia

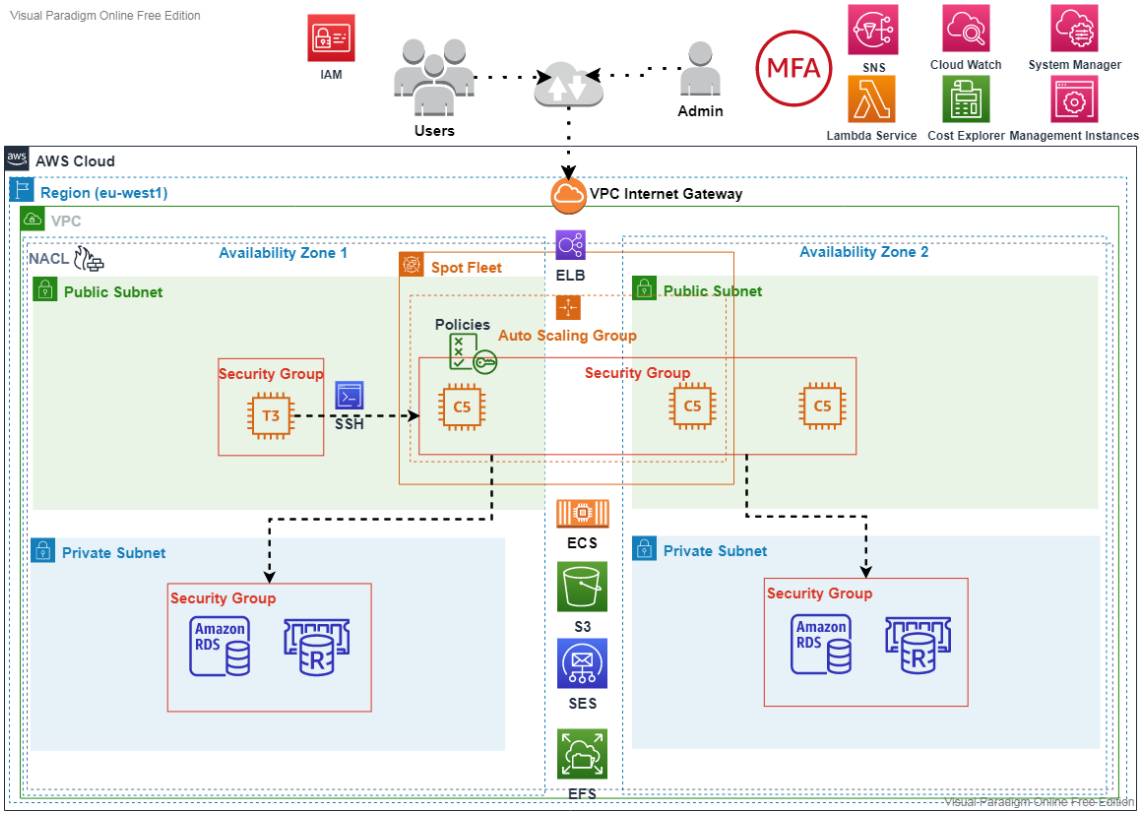


Como se puede ver en el diagrama de la **Figura 30**, este entorno cuenta con una *Virtual Private Cloud* (VPC) protegida a través de una *Network Access Control List* (NACL). En esta VPC se encuentra ubicada una sola zona de disponibilidad, por lo que los recursos no se encuentran duplicados, ya que no es necesario disponer ni de alta disponibilidad, ni de tolerancia a fallos. Contará con dos subredes (*subnets*), las cuales están diferenciadas en pública y privada. Por lo que los servicios que se localicen en la pública podrán ser accesibles a través de internet, mientras que en la privada no lo serán. Habrá una sola instancia EC2 (*Elastic Compute Cloud*) que contendrá el servidor web, la memoria caché y la tecnología PHP. Y accederá a la base de datos localizada en la *Relational Data Service* (RDS) localizada en la subred privada. Por lo que ambas estarán protegidas en cuanto a *firewall* a partir de sus respectivos grupos de seguridad o *Security Groups* (SG) que contendrán las políticas de entrada y salida. Este entorno será monitorizado de forma continua a través del servicio *CloudWatch* para conocer sus prestaciones y estudiar mejoras en una versión definitiva, y por otro lado, para ahorrar costes, las instancias EC2 y RDS no estarán operativas un periodo de 24/7 sino que se encenderán y apagarán de forma automática a través de reglas definidas en el servicio de Lambda. La instancia EC2 obtendrá los datos de los sensores a través de una sincronización con un *bucket* ubicado en el *Simple Storage Service* (S3), y cuyo acceso es realizado a través de credenciales. Y el envío de correos se realizará a través del servicio de AWS conocido como *Simple Email Service* (SES), este servicio está acotado en un modo *SandBox* por seguridad. El acceso a esta infraestructura desplegada lo podrá realizar el administrador, quien también tiene el rol de usuario del sistema, a través del servicio *Identity Access Management* (IAM) por medio de claves de tipo usuario y contraseña y un doble factor de autenticación, por lo que el acceso se seguro.

Una vez que el sistema se prueba en el entorno pre-productivo, y funciona de forma correcta, se puede pasar al entorno productivo.

Este entorno está diseñado para brindar al sistema de una alta disponibilidad y una óptima tolerancia a fallos, pues cuenta con dos zonas de disponibilidad donde los recursos se encuentran replicados. El diagrama de arquitectura de este entorno se puede visualizar en la **Figura 31**.

Figura 31. *Diagrama de arquitectura del sistema en el entorno productivo.* Elaboración Propia

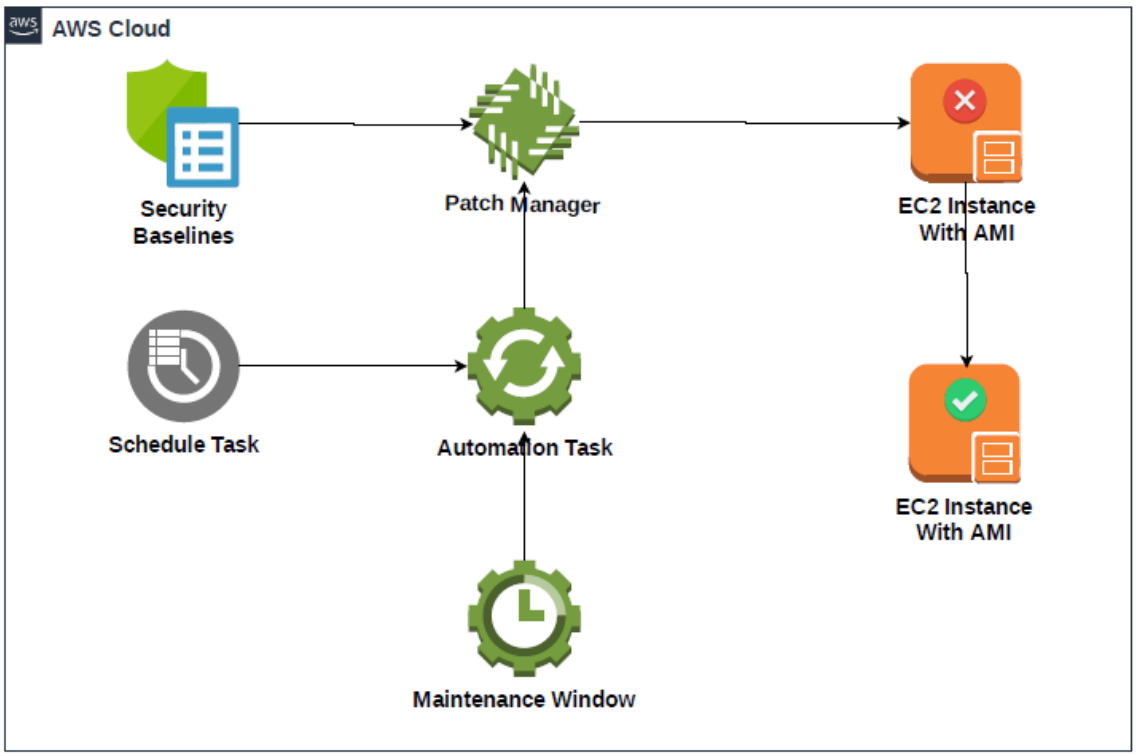


Adicionalmente, cuenta con más recursos como pueden ser balanceadores de carga de tipo *Elastic Load Balancer* (ELB) que distribuyen las peticiones al sistema de forma que, si una instancia EC2 se encuentra con mayor carga que otra, el tráfico irá a la que menos carga tenga. O un servicio de Redis, para el almacenamiento de la sesión en caché, por lo que el almacenamiento de esta memoria pasará de estar integrado en el propio servidor web a ser un servicio autogestionado por AWS conocido como ElastiCache. Monitorización continua de los sistemas y con mayor precisión, por lo que, si en algún momento el sistema se ve alterado, se enviará un aviso al administrador a través del servicio de notificaciones conocido como *Simple Notification Service* (SNS) para que el administrador pueda solventar el problema, así mismo, este servicio también permite avisar si, por ejemplo, pasa de un umbral de costes fijado. Por otro lado, las instancias web contarán con una ruta montada en el sistema de ficheros en red de AWS conocido como *Elastic File System* (EFS) de esta forma, todas las instancias que vayan escalando por necesidad de recursos gracias al *Auto Scaling Group* (ASG), contarán con el código PHP del sistema además de la misma configuración de PHP, NGINX, certificados, etc. Estas instancias web se obtendrán a través de un conjunto de instancias alojadas en una flota de instancias Spot, de esta forma se pueden utilizar instancias en función de filtros

(precios, características hardware de la máquina, CPU, tipos, etc.), así mismo las instancias serán gestionadas a través de una instancia bastión que será de menor rango que estas, para así gestionar las instancias web a través de esta y dinamizar por ejemplo procesos de actualización, o reinicio de servicios.

Adicionalmente, este entorno contará con servicios que sean encargados de actualizar de forma automática las máquinas y aplicar parches de seguridad, este proceso se puede visualizar de forma gráfica en la **Figura 32**. Consiste en la configuración de una ventana de mantenimiento que ejecutará una tarea automática programada la cual cargará de unas plantillas de seguridad las actualizaciones y parches a aplicar en la instancia EC2 para actualizarla.

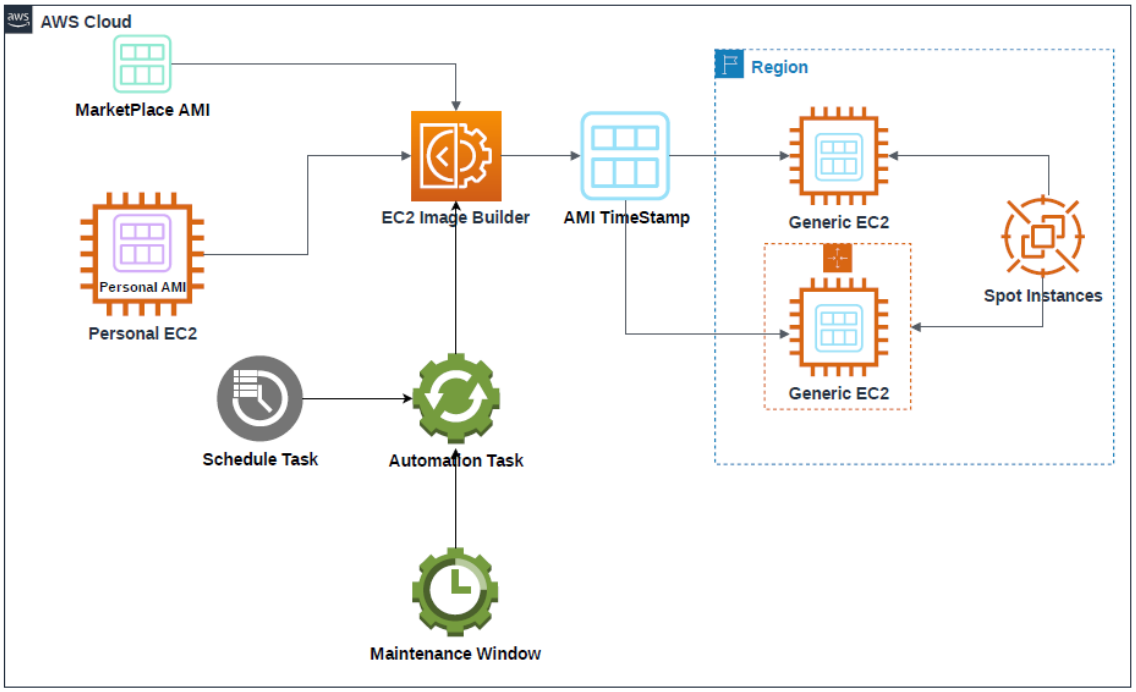
Figura 32. *Proceso de aplicación de parches y políticas de seguridad automático en AWS.* Elaboración Propia



O realizar imágenes de las máquinas, en AWS se conocen como *Amazon Machine Image* (AMI) que contendrán toda la información y configuración de esta para así poder actualizar de forma automática el ASG y que utilice imágenes actualizadas, este proceso se puede visualizar de forma gráfica en la **Figura 33**. Consiste en la programación de una ventana de mantenimiento que de nuevo ejecutará una tarea automática programada y realizará una copia de una AMI personal ya existente o una del *MarketPlace* de AMIs, se le especificará un sello de tiempo para tenerlas ordenadas y se aplicará a la configuración

de la plantilla del ASG para que las nuevas máquinas que se levanten tengan esta nueva imagen de forma automática.

Figura 33. *Proceso de creación de AMI y actualización del ASG de forma automática en AWS.* Elaboración Propia



Además, complementando a la aplicación principal, se dispondrá de la utilización de microservicios, estos son desplegados en imágenes de Docker a través del servicio de AWS conocido como ECS, por lo que la aplicación podrá hacer uso de estos a través de end-points los cuales hacen referencia a estos contenedores. Un ejemplo implementado es el antivirus ClamAV, que permite el escaneo de ficheros para comprobar si son maliciosos o no a través de la huella de diferentes virus de los cuales se tienen registros, por lo que si detecta la presencia de software malicioso no los procesará.

Todos estos servicios adicionales y gestionados por AWS consiguen incrementar la eficiencia del sistema. Como los servicios son mayores y con mejores prestaciones que en el entorno pre-productivo el coste también será mayor, por este motivo este entorno solo se implementará cuando el sistema haya pasado todas las pruebas oportunas y esté preparado para ser utilizado en el entorno productivo.

Todos los servicios empleados, su funcionamiento y características se disponen en el **Anexo III** de la memoria para ser consultados.

6. Conclusiones

Tras la implementación y pruebas realizadas en este proyecto, se extraen numerosas conclusiones tanto a nivel personal como técnico.

Este proyecto ha permitido la adquisición de nuevos conocimientos y sobre todo ha permitido experimentar los roles más habituales de integrantes de un equipo de desarrollo, es decir, desarrollo de *frontend*, desarrollo de *backend*, diseño de *UX* y análisis heurístico de la interfaz web, aseguramiento de la calidad y pruebas, gestión de los recursos (tiempo y costes), etc. Y, finalmente, ha permitido la realización exitosa de los objetivos técnicos del sistema mencionados al comienzo de este documento, los cuales son los siguientes:

- **Gestión Agrícola**: Permite monitorizar y tener un registro histórico de los elementos influyentes en el cultivo de productos agrícolas (agua, temperaturas, sulfatos y minerales) para así aumentar el rendimiento de los cultivos y reducir los costes asociados al cuidado de estos.
- **Gestión Ganadera**: Utilizado para la identificación y control de las cabezas de ganado en el sistema además de su localización GPS para tenerlas controladas y en caso de que se desubicaran esto no suponga una pérdida en costes.
- **Gestión de Recursos Humanos**: Comprende las acciones correspondientes a la adición de nuevos usuarios que puedan acceder al sistema, diferenciados en dos roles, los administradores que podrán ejercer un control sobre los dispositivos IoT y tener un control sobre los proveedores. Estos últimos para evitar posibles brotes derivados de la crisis sanitaria ocasionada por el Sars-CoV-2, pueden escanear un código QR para tener un control de cuando han accedido a la finca.
- **Gestión del Sistema**: Se basa mantener una correcta sincronización entre los elementos representados en la plataforma y los obtenidos por los sensores IoT, a la par que brindar de una correcta experiencia de usuario y mantener la seguridad y disponibilidad óptimas. Así mismo se ha conseguido una gestión óptima de los archivos y directorios que se manejan en el sistema

Algunos de los conocimientos que se han adquirido en el desarrollo de este proyecto es por ejemplo el manejo y correcto desempeño de un *framework* de desarrollo (la primera versión fue realizada en PHP nativo y sin ningún patrón de diseño), como es el caso de Yii2 utilizado para el lenguaje PHP junto con el patrón MVC, lo que también ha implicado en un primer estudio de este *framework*, sus características y el uso de

bibliotecas o extensiones de terceros. Por otro lado, también se ha enriquecido la resolución de problemas que se iban presentando a lo largo de la elaboración del sistema, por lo que ha sido necesario hacer búsquedas de forma concreta a los problemas surgidos y como resolverlos, cabe destacar que todas las búsquedas se realizaban en inglés, por lo que también se entrenaba el uso de este idioma.

Además, también se ha especializado en el proveedor de *cloud* de AWS, aprendiendo y añadiendo nuevos servicios como pueden ser el envío de correos, la gestión de caché y sesión, la alta disponibilidad, el balanceo de cargas, la notificación de problemas en el sistema o la realización de copias de seguridad y parches de seguridad en las máquinas. Todos estos servicios gestionados por AWS pero siendo necesario un aprendizaje sobre el correcto uso de estos. A lo que hay que sumarle que la mayor parte de esta documentación era de nuevo en inglés.

Por otro lado, se han desarrollado habilidades con la parte más electrónica de este proyecto, ya que se ha tratado de realizar un sistema de envío de datos a través de un protocolo que no suponga un coste excesivo, a través de una correcta gestión de los sensores y la construcción de un prototipo básico de cómo podría ser un sensor de temperaturas y humedad. Para ello ha sido necesario conocer la estructura hardware de los dispositivos involucrados, así como su rango de voltajes, su amperaje y calibrado para obtener unos datos correctos.

Finalmente, todo lo anterior ha supuesto una gestión en los tiempos de desarrollo del proyecto, ya que se han utilizado tecnologías novedosas para mí y sin un aprendizaje previo, como puede ser el desarrollo con un *framework* o el despliegue de una aplicación con alta disponibilidad y tolerancia a fallos en un proveedor de *cloud*. A lo que sumado al diseño de un sistema apto para usuarios que no cuenten con una previa experiencia en sistemas TI (análisis heurístico y carga cognitiva en el usuario). Han supuesto un reto para mí, no obstante, considero que todo este camino ha merecido la pena, ya que he podido meterme en la piel de todos los integrantes de un equipo de desarrollo de software.

7. Líneas de trabajo futuras

Como líneas de investigación futuras para nuevas integraciones en el sistema a corto o medio plazo se plantean las siguientes:

- Adición de nuevos microservicios, desplegados en contenedores de Docker en el servicio gestionado por AWS denominado Elastic Container Service (ECS).
- El bróker físico desaparecerá, de forma que sean los propios dispositivos IoT los que transfieran los datos a través de redes como 4G o 5G a un servidor en *cloud* y este sea el nuevo bróker.
- Utilización de un concentrador en el protocolo MQTT.

Y como implementación a largo plazo, se encuentran las siguientes:

- Desarrollo de una aplicación nativa en Android y IOS.
- Adición de nuevas funcionalidades a la gestión ganadera como pueden ser la obtención de datos biométricos de las cabezas de ganado (hidratación, fecha óptima para ser ordeñada, fecha óptima de consumo, etc.)
- Evolución a un sistema completamente *serverless*, donde desaparezcan las instancias web y permitir que sea el servicio Lambda de AWS quien gestione el sistema a través de servicios autogestionados.

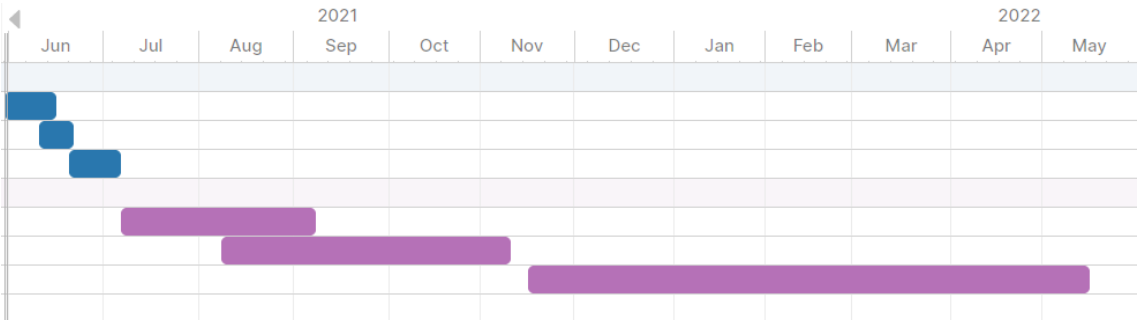
A continuación, a través de un Diagrama de Gantt con una estimación teórica para el desarrollo de las nuevas integraciones, siguiendo con la misma metodología se estima que el tiempo para la conclusión de estas nuevas integraciones dure 274 días, como se puede ver en la siguiente figura:

Figura 34. *Estimación temporal para las nuevas integraciones.* Elaboración propia

Activity	Start	End	Days
Integraciones a corto / medio plazo	**31-05-21**	**06-07-21**	**32.0**
Adición de nuevos microservicios	31-05-21	15-06-21	12.0
Implementación de broker on cloud	10-06-21	21-06-21	8.0
Utilización de concentrador	21-06-21	06-07-21	12.0
Integraciones a futuro plazo	**07-07-21**	**16-05-22**	**242.0**
Desarrollo de aplicación nativa Android y IOS	07-07-21	07-09-21	45.0
Adición de nuevas funcionalidades ganado	09-08-21	09-11-21	67.0
Evolución a un sistema serverless	16-11-21	16-05-22	130.0
			274.0

En el cual las integraciones a corto o medio plazo se desarrollarían en poco más de un mes. Y finalmente el Diagrama de Gantt:

Figura 35. *Diagrama de Gantt para las nuevas integraciones.* Elaboración propia



8. Bibliografía

AliOs (6 de noviembre de 2019). Raspberry Pi 4 GPIO Pinout. *Key to Smart*. https://keytosmart.com/single-board-computers/raspberry-pi-4-gpio-pinout/

Aprendiendo Arduino (s.f.). *Práctica 4: instalar, configurar y securizar Mosquitto y Node-RED en Raspberry Pi*. https://aprendiendoarduino.wordpress.com/tag/securizar-mosquitto/

AWS (s.f.). *Consola de administración de AWS, guía de introducción* [Archivo PDF]. https://docs.aws.amazon.com/es_es/awsconsolehelpdocs/latest/gsg/console-help-gsg.pdf

Bernárdez, B. y Durán, A. (octubre de 2000). *Metodología para la Elicitación de Requisitos de Sistemas Software*. Universidad de Sevilla. http://www.lsi.us.es/docs/informes/lsi-2000-10.pdf

CaraballoMaestre (6 de mayo de 2011). ¿Por qué Yii Framework? *Ingeniería de Software*. http://caraballomaestre.blogspot.com/2011/05/por-que-yii-framework.html

Cockroach Labs (12 de enero de 2021). *2021 Cloud Report, Cockroach Labs*. https://resources.cockroachlabs.com/guides/2021-cloud-report

David Estevez (s.f). *Usando etiquetas.* https://david-estevez.gitbooks.io/the-git-the-bad-and-the-ugly/content/es/usando-etiquetas.html

Del Valle, L. (2021). WifiManager librería para configurar red WiFi de un ESP8266. *Programarfacil*. https://programarfacil.com/esp8266/wifimanager-configura-wifi-esp8266/

Durán, A. (25 de noviembre de 2004). Herramienta REM. *Universidad de Sevilla*. http://www.lsi.us.es/descargas/descarga_programas.php?id=3

Escuela Técnica Superior de Ingeniería informática (s.f.). *Diseño de la capa de datos. Patrón Dao* [Archivo PDF]. http://www.lsi.us.es/docencia/get.php?id=1326

ESP826 Community Forum (2015). *Coverting ipaddress to string*. https://www.esp8266.com/viewtopic.php?p=38877

Espruino (2017). *DHT22/AM230x/RHT0x Temperature and RH Sensor*. https://www.espruino.com/DHT22

Evans, B. (7 de noviembre de 2017). The Top 5 Cloud- Computing Vendors: #1 Microsoft, #2 Amazon, #3 IBM, #4 Salesforce, #5 SAP. *Forbes*. https://www.forbes.com/sites/bobevans1/2017/11/07/the-top-5-cloud-computing-vendors-1-microsoft-2-amazon-3-ibm-4-salesforce-5-sap/?sh=286e00f06f2e

FAO (12 de octubre de 2009). La agricultura mundial en la perspectiva del año 2050. http://www.fao.org/fileadmin/templates/wsfs/docs/Issues_papers/Issues_papers_SP/La_agricultura_mundial.pdf

Flores, F. (22 de marzo de 2021). Cloud Computing, tipos de nubes, servicios y proveedores. Open Webinars. https://openwebinars.net/blog/tipos-de-cloud-computing/

Gao, F., Intizar, M., Kamilarism A. y Prenafeta, F. X. (2021). Agri-IoT: A semantic framework for internet of things-enabled smart farming applications. IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/7845467

García Holgado, A. y García Peñalvo, F. J. (2018). *Ingeniería de Software I* [Archivo PDF]. https://docplayer.es/123234799-Requisitos-ingenieria-de-software-i-francisco-jose-garcia-penalvo-alicia-garcia-holgado.html

Geerling, J. (2021). Ansible Role: MySQL. *GitHub*. https://github.com/geerlingguy/ansible-role-mysql

Geerling, J. (2021). Ansible Role: Nginx. *GitHub*. https://github.com/geerlingguy/ansible-role-nginx

Geerling, J. (2021). Ansible Role: PHP. *GitHub.* https://github.com/geerlingguy/ansible-role-php

GitHub (2021). *Ubuntu 18.04 CIS Benchmark for Ansible*. https://github.com/SecurityVoyage/ubuntu-18.04-cis-benchmark-for-ansible

Gondechawar, N. y Kawitkarm, R. S. (2016). IoT based Smart Agriculture. *International journal of advanced research in computer and communication engineering, 5*, 838- 842. DOI 10.17148/IJARCCE.2016.56188. http://www.kresttechnology.com/krest-academic-projects/krest-major-projects/ECE/BTech%20%20Major%20ECE%20EMBEDDED%202016-17/Btech%20ECE%20Embedded%20Major%20BP%202016-17/3.%20Automated%20Irrigation%20System%20In%20Agriculture.pdf

González, L. (17 de agosto de 2018). Los 10 lenguajes de programación más populares. *Dinahosting*. https://dinahosting.com/blog/los-10-lenguajes-de-programacion-mas-usados/

Gracia, M. (2021). IoT- Internet of Things. *Deloitte*. https://www2.deloitte.com/es/es/pages/technology/articles/IoT-internet-of-things.html

Gran Vía Venue (7 de octubre de 2021). IoT Transforming the future of agricultura. *IoT Solutions Word Congress*. https://www.iotsworldcongress.com/iot-transforming-the-future-of-agriculture/#:~:text=IoT%20in%20agriculture%20technologies%20comprise,connectivity%2C%20software%20and%20IT%20services.&text=IoT%20smart%20farming%20solutions%20is,%2C%20crop%20health%2C%20etc

Hosting (19 de octubre de 2017). Principales lenguajes de programación para el desarrollo web. *Piensa Solutions*. https://www.piensasolutions.com/blog/principales-lenguajes-programacion-web/

IAT (2020). *IOT en agricultura. Cultivos y dispositivos inteligentes*. https://iat.es/tecnologias/internet-de-las-cosas-iot/agricultura/

IBM Cloud Education (10 de octubre de 2018). *IaaS frente a PaaS frente a SaaS*. https://www.ibm.com/es-es/cloud/learn/iaas-paas-saas

Isaac (s.f.). MQTT: un protocolo abierto de red y su importancia en el IoT. *Hardware libre*. https://www.hwlibre.com/mqtt/

Jesuïtes Educació (1 de noviembre de 2020). *Los 5 mejores proveedores de Cloud Computing del 2020*. https://fp.uoc.fje.edu/blog/los-5-mejores-proveedores-en-cloud-computing/

Jones, E. (28 de enero de 2021). Google Cloud vs AWS en 2021 (Comparación de los gigantes). *Kinsta*. https://kinsta.com/es/blog/google-cloud-vs-aws/

Jones, E. (5 de marzo de 2021). AWS vs Azure en 2021 (Comparación de los gigantes de la computación en la nube). *Kinsta*. https://kinsta.com/es/blog/aws-vs-azure/

Kadam, A. (2018). Visión general del mercado de IoT en la agricultura: 2025. *Allied Market Research*. https://www.alliedmarketresearch.com/internet-of-things-iot-in-agriculture-market

Kanbanize (2021). *Qué es un tablero Kanban: fundamentos*. https://kanbanize.com/es/recursos-de-kanban/primeros-pasos/que-es-tablero-kanban

Línea de Código (19 de marzo de 2013). *Patrón Abstract Factory*. http://lineadecodigo.com/patrones/patron-abstract-factory/

Linkeit (2020). *Infraestructuras On Promise vs Cloud*. https://www.linkeit.com/es/blog/infraestructuras-on-premise-vs-cloud

Luis Llamas (1 de junio de 2018). Nodemcu, la popular plaza de desarrollo con ESP8266. https://www.luisllamas.es/esp8266-nodemcu/

Luis Llamas (13 de diciembre de 2016). Medir caudal y consumo de agua con Arduino y caudalímetro. https://www.luisllamas.es/caudal-consumo-de-agua-con-arduino-y-caudalimetro/

Luis Llamas (13 de febrero de 2016). *Detector de lluvia con Arduino y sensor FC-37 o YL-83*. *https://www.luisllamas.es/arduino-lluvia/*

Luis Llamas (17 de abril de 2019). *¿Qué es MQTT? Su importancia como protocolo IoT*. https://www.luisllamas.es/que-es-mqtt-su-importancia-como-protocolo-iot/

Luis Llamas (19 de enero de 2016). Medir la humedad del suelo con Arduino e hidrómetro FC-28. https://www.luisllamas.es/arduino-humedad-suelo-fc-28/

Luis Llamas (21 octubre de 2016). Detector de gases con Arduino y la familia de sensores MQ. https://www.luisllamas.es/arduino-detector-gas-mq/

Luis Llamas (27 de septiembre de 2016). Localización GPS con Arduino y los módulos GPS NEO-6. https://www.luisllamas.es/localizacion-gps-con-arduino-y-los-modulos-gps-neo-6/

Luis Llamas (29 de marzo de 2016). Medir temperatura y humedad con Arduino y sensor DHT11- DHT22. https://www.luisllamas.es/arduino-dht11-dht22/

Martínez, D. (2021). DevOps una metodología que trabaja en el desarrollo de software ágiles. *Deloitte.* https://www2.deloitte.com/es/es/pages/technology/articles/devops-metodologia-desarrollo-software.html

Martínez, L. (19 de octubre de 2020). ¿Qué es Adobe XD y para qué sirve? *Blog. Hubspot*. https://blog.hubspot.es/marketing/para-que-sirve-adobe-xd

Microsoft (2021). *Microsoft compliance offerings.* https://docs.microsoft.com/en-us/compliance/regulatory/offering-home

Microsoft Azure (2021). *¿Qué es la nube pública, privada e híbrida?* https://azure.microsoft.com/es-es/overview/what-are-private-public-hybrid-clouds/#deployment-options

Mora, A. (3 de marzo de 2021). Los mejores navegadores web de 2021. *PCWorld*. https://www.pcworld.es/mejores-productos/internet/mejores-navegadores-web-3672988/#:~:text=Google%20Chrome%20es%2C%20de%20lejos,bajo%20de%20los%20%C3%BAltimos%20a%C3%B1os.

Noguera, B. (s.f.). Raspberry Pi: qué es, características y precios. *Culturación. https://culturacion.com/raspberry-pi-que-es-caracteristicas-y-precios/*

Packagist (s.f.). V*isor de formato de documento portátil (PDF) de extensión Yii2 pdf.js.*. https://packagist.org/packages/yii2assets/yii2-pdfjs

Pelado Nerd (2021). *Docker 2021- de novato a pro (Curso completo)* [Archivo de Vídeo]. Youtube https://www.youtube.com/watch?v=CV_Uf3Dq-EU

PHP Benchmarks (2021). *Perfomance tolos*. http://www.phpbenchmarks.com/en/benchmark/yii/2.0

Programador Clic (2021). *Patrón compuesto de patrones de diseño (Patrón compuesto)*. https://programmerclick.com/article/4752342119/

Raúl Ávila (11 de abril de 2015). *El patrón Composite en la práctica*. http://raulavila.com/2015/04/patron-composite/

Red Hat (2021). *Drive automation across open hybrid cloud deployments*. https://www.ansible.com/

Redacción (2014). Cómo descargar e instalar Arduino y el IDE en nuestro ordenador. *Descubre Arduino*. https://descubrearduino.com/instalar-arduino/

Redes Zone (s.f.). *MobaXterm: Terminal para Windows 10 con cliente SSH y SFTP*. https://www.redeszone.net/analisis/software/mobaxterm-terminal-windows/

Redes Zone (s.f.). *Qualys SSl Labs actualiza sus test para comprobar la seguridad de tu página web*. https://www.redeszone.net/2017/01/15/qualys-ssl-labs-actualiza-sus-test-para-comprobar-la-seguridad-de-tu-pagina-web/

Redes Zone (s.f.). *Vagrant: instalación, configuración y ejemplos de uso de esta herramienta*. https://www.redeszone.net/tutoriales/servidores/vagrant-instalacion-configuracion-ejemplos/

RedHat (2021). *¿Qué es el cloud computing? https://www.redhat.com/es/topics/cloud*

RedHat (2021). *¿Qué es la arquitectura en la nube?* https://www.redhat.com/es/topics/cloud-computing/what-is-cloud-architecture#:~:text=La%20arquitectura%20de%20nube%20constituye,recursos%20escalables%20en%20una%20red.

Refactoring.Guru (2021). *Abstract Factory*. https://refactoring.guru/es/design-patterns/abstract-factory

Refactoring.Guru (2021). *Composite*. https://refactoring.guru/es/design-patterns/composite

Reno, N. V. (29 de octubre de 2020). La tasa de crecimiento del mercado de la nube aumenta a medida que Amazon y Microsoft consolidan su liderazgo. *Synergy Research Group*. https://www.srgresearch.com/articles/cloud-market-growth-rate-nudges-amazon-and-microsoft-solidify-leadership

Robledano, A. (11 de junio de 2019). Qué es la programación orientada a objetos. *Open Webinars.* https://openwebinars.net/blog/que-es-la-programacion-orientada-objetos/

Ros, I. (10 de enero de 2019). Estos son los lenguajes de programación más populares en 2019. *Muycomputerpro*. https://www.muycomputerpro.com/2019/01/10/lenguajes-de-programacion-mas-populares

Santander Universidades (21 de diciembre de 2020). Metodologías de desarrollo de software: ¿qué son? *Santander*. https://blog.becas-santander.com/es/metodologias-desarrollo-software.html

Servicio de Informática ASP.NET MVC 3 Framework (2021). Modelo Vista Controlador. *Universidad de Alicante*. https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html

Soft Zone (s.f.). *Hacer una página web: lenguaje de programación más usados*. https://www.softzone.es/programas/lenguajes/lenguajes-programacion-web/

Soft Zone (s.f.). *Visual Studio Code: editor de código abierto para programar*. https://www.softzone.es/programas/utilidades/visual-studio-code/

Statista Research Department (5 de febrero de 2021). *IoT in agricultura market size globally 2018/2023*. https://www.statista.com/statistics/766793/global-iot-in-agriculture-market-size/

Tébar, E. (13 de febrero de 2020). Frameworks en el desarrollo web: las mejores prácticas para tu negocio online. *We are marketing*. https://www.wearemarketing.com/es/blog/frameworks-en-el-desarrollo-web-las-mejores-practicas-para-tu-negocio-online.html

Terraform (s.f.). *Write, Plan, Apply*. https://www.terraform.io/

Tortoise SVN (2021). *About Tortoise SVN*. https://tortoisesvn.net/

Vagrant Cloud (2021). *Discover Vagrant Boxes*. https://app.vagrantup.com/boxes/search

Wikipedia (21 de enero de 2021). *KeePass*. https://es.wikipedia.org/wiki/KeePass

Wikipedia (24 de enero de 2020). *Fritzing*. https://es.wikipedia.org/wiki/Fritzing

Wikipedia (29 de julio de 2020). *Gitlab*. https://es.wikipedia.org/wiki/GitLab

Yiiframework (2021). *Modelo-Visa-Controlador (Model-View-Controller MVC)*. https://www.yiiframework.com/doc/guide/1.1/es/basics.mvc