



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Máster Online en Ingeniería Informática



**TFM del Máster Universitario
Online en Ingeniería Informática**

**Sistema prototípico para la
monitorización de sistemas de
riego**



Presentado por David Álvarez Castro
en la Universidad de Burgos a 11 de
septiembre de 2022

Tutor: Carlos Cambra Baseca
Cotutor: Daniel Urda Muñoz



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR

Máster Online en Ingeniería Informática



D. Carlos Cambra y D. Daniel Urda, profesores del departamento de Ingeniería Informática, área de Ciencia de la Computación e Inteligencia Artificial.

Expone:

Que el alumno D. David Álvarez Castro, con DNI 45848090-M, ha realizado el Trabajo final de Máster Online en Ingeniería Informática titulado **Sistema prototípico para la monitorización de sistemas de riego**.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 11 de septiembre de 2022

Vº. Bº. del Tutor:

D. Carlos Cambra Baseca

Vº. Bº. del Co-tutor:

D. Daniel Urda Muñoz

Resumen

Es un hecho que el impacto que están teniendo en los últimos años las Tecnologías de la Información en todos los ámbitos de nuestra vida ha permitido cambiar la forma de interactuar y percibir nuestro entorno. La agricultura es un claro ejemplo de los grandes beneficios que se puede obtener al aplicar este tipo de tecnología en este campo; con un impacto determinante en el ámbito de la agricultura de precisión y la sensorización de los campos de cultivo. Todo ello enmarcado en unas circunstancias actuales muy críticas relativas a la gestión óptima del agua derivadas del cambio climático.

Se ha diseñado y desarrollado un prototipo de sistema que permite gestionar un sistema de riego mediante el uso de un sistema empotrado (actuador), a partir de los datos medioambientales obtenidos mediante el uso de sensores conectados a otro sistema empotrado (controlador). El sistema proporciona una aplicación web en la cual el usuario final puede monitorizar los valores medioambientales, como temperatura o humedad, obtenidos por cada uno de los controladores registrados en el sistema y los períodos de activación de los actuadores.

Para finalizar, se ha optimizado el despliegue de los diferentes servicios utilizando una estructura de gestión basada en contenedores de Docker: un contenedor para el despliegue de la web de la aplicación, uno para el despliegue del servicio de mensajes asíncronos, otros dos para el despliegue de las bases de datos, dos contenedores independientes para el despliegue de los microservicios encargados de la comunicación con los microcontroladores y, por último, dos contenedores dedicados al despliegue de las *Apis REST* de autenticación y gestión. La comunicación entre los diferentes elementos del sistema se lleva a cabo mediante protocolos de comunicación asíncrona y peticiones *HTTP*.

Descriptores

Riego, agricultura, sensores, IoT, Flask, MQTT, microservicios, Arduino y Raspberry.

Abstract

The emergence of Information and Communication Technologies in recent times has revolutionized the form of exchange and perception of the environment. A very clear example of how to get the benefits of applying this kind of technology in this field of study is agriculture. There are many fields where we can apply this technological knowledge and get a very positive impact, such as precision irrigation and crop field *sensorization*; even more nowadays considering the current circumstances about optimal water management derived from climate change.

A system prototype for an irrigation management system embedded system-based (*a.k.a.* actuator) has been designed and developed by using environmental data retrieved from a set of sensors connected to another embedded system (*a.k.a.* controller) as the main source to decide whenever to activate or not the actuator system. This project provides a web application where the end user can analyze environmental data from sensors, like air temperature or humidity, and the actuator's activation period by graph elements.

Finally, a structure of Docker container has been designed and deployed to separate each service from the application: web application, asynchronous system broker, databases, microservices for the communication between server-microcontrollers, authentication API REST, and management API REST. These containers have been communicated by using HTTP requests and asynchronous events protocols.

Keywords

Irrigation, agriculture, sensors, IoT, Flask, MQTT, microservices, Arduino and Raspberry.

Índice general

Índice general	iii
Índice de figuras	vi
Índice de tablas	viii
1 Introducción	1
1.1. Contexto	1
1.2. Motivación	2
1.3. Problema	3
1.4. Organización de la memoria	4
2 Objetivos del proyecto	7
2.1. Objetivo General	7
2.2. Objetivos intermedios	8
3 Conceptos teóricos	11
3.1. Riego de precisión	11
3.2. Bases de datos no relacionales	13
3.3. Lógica difusa	14
3.4. Comunicación asíncrona	15
3.5. Desarrollo de sistemas embebidos	16
4 Técnicas y herramientas	19
4.1. Metodologías Ágiles	19
4.2. Despliegue basado en contenedores	20
4.3. Sistemas empotrados	21

4.4. Microservicios	23
4.5. Otras herramientas	23
5 Aspectos relevantes del desarrollo del proyecto	25
5.1. Inicio del proyecto	25
5.2. Formación	25
5.3. Arquitectura	26
5.4. Diseño de las bases de datos	28
5.5. Servicios y Microservicios	29
5.6. Comunicación asíncrona	31
5.7. Aplicación Web	33
5.8. <i>Testing</i>	38
5.9. Despliegue en producción	38
6 Trabajos relacionados	41
6.1. Gardena	41
6.2. Libelium	42
6.3. WheaterTRAK	43
6.4. Sativum	44
6.5. Aplicando modelos basados en Inteligencia Artificial	45
7 Conclusiones y Líneas de trabajo futuras	47
7.1. Objetivos cumplidos	47
7.2. Competencias adquiridas	49
7.3. Problemas y limitaciones del sistema	50
7.4. Mejoras y Trabajo futuro	51
Apéndice A Plan de Proyecto Software	53
A.1. Introducción	53
A.2. Planificación temporal	53
A.3. Viabilidad económica	66
A.4. Viabilidad legal	68
Apéndice B Especificación de Requisitos	69
B.1. Introducción	69
B.2. Objetivos generales	69
B.3. Especificación de requisitos	69
Apéndice C Especificación de diseño	75
C.1. Diagrama de Casos de Uso	75
C.2. Diseño de datos	81
C.3. Diagrama de Arquitectura	85

C.4. Diagramas de Secuencia	87
C.5. Diagrama de Despliegue	90
C.6. Diseño de la Interfaz	91
C.7. Diseño circuito eléctrico	94
Apéndice D Documentación técnica de programación	97
D.1. Introducción	97
D.2. Estructura de directorios	98
D.3. Manual del programador	103
D.4. Compilación, instalación y ejecución del proyecto	105
D.5. Pruebas del sistema	110
Apéndice E Documentación de usuario	119
E.1. Introducción	119
E.2. Requisitos de usuarios	119
E.3. Instalación	120
E.4. Manual del usuario	120
Bibliografía	127

Índice de figuras

3.1. Evapotranspiración del cultivo	12
3.2. Ejemplo de documentos	13
3.3. Función de pertenencia de <i>alto</i> definido mediante conjuntos clásicos y borrosos	14
3.4. Función de pertenencia de <i>temperatura, humedad y lluvia</i>	15
3.5. Tipos de señales	17
3.6. Convertidor <i>analog-to-digital</i>	17
4.1. Diagrama con el proceso Scrum	20
5.1. Arquitectura del sistema	27
5.2. Arquitectura de comunicación	32
5.3. Mapa Web de la Aplicación	34
5.4. Boceto: Página de Inicio	34
5.5. Boceto: Visualización de histórico de datos	35
5.6. Aplicación web: Página de Inicio	36
5.7. Aplicación web: Gestión de Zonas	37
5.8. Aplicación web: Monitorización de sensores	37
6.1. Productos sensorizados de Gardena	42
6.2. Productos sensorizados de Gardena (II)	42
6.3. Arquitectura Libelium Cloud	43
6.4. Dispositivos de riego de WheaterTRAK	44
6.5. Controladores de riego de WheaterTRAK	44
6.6. Aplicación Web de Sativum	45
A.1. Planificación temporal del proyecto	65
C.1. Diagrama de Casos de Uso	76

C.2. Diagrama Entidad-Relación Controladores	81
C.3. Diagrama Entidad-Relación Usuarios	82
C.4. Diagrama Relacional Controladores	82
C.5. Diagrama Relacional Usuarios	83
C.6. Colecciones Histórico	84
C.7. Diagrama de arquitectura del sistema	85
C.8. Módulos del sistema	86
C.9. Diagrama de Secuencia: Visualizar histórico de sensores	87
C.10. Diagrama de Secuencia: Gestión de controladores	88
C.11. Diagrama de Secuencia: Controlador Sensores	89
C.12. Diagrama de Despliegue	90
C.13. Boceto: Página de Login	91
C.14. Boceto: Página de Inicio	92
C.15. Boceto: Gestión de Zonas	92
C.16. Boceto: Gestión de Controladores	93
C.17. Boceto: Visualización de histórico de datos	93
C.18. Diagrama de conexiones del Controlador	94
C.19. Diagrama de conexiones del Actuador	95
C.20. Diagrama de conexiones del Actuador (II): led	95
D.1. Prueba web: Credenciales erróneas	113
D.2. Prueba web: Listado de zonas	113
D.3. Prueba web: Nueva zona	114
D.4. Prueba web: Zona actualizada	115
D.5. Prueba web: Zona eliminada	115
D.6. Prueba web: Nuevo controlador	116
D.7. Prueba web: Histórico datos sensores	117
D.8. Prueba web: Histórico datos activación	118
E.1. Pantalla de Login	121
E.2. Pantalla de Inicio	122
E.3. Pantalla de Inicio para los diferentes usuarios	123
E.4. Listado de controladores	123
E.5. Modales de gestión de zonas	124
E.6. Modales de gestión de controladores	125
E.7. Vista del histórico de datos de los sensores	125
E.8. Vista del histórico de activación	126

Índice de tablas

A.1. Tabla de costes	68
C.1. Caso de Uso Login	76
C.2. Caso de Uso Logout	77
C.3. Caso de Uso Consultar Zona	77
C.4. Caso de Uso Estado Actual Sensores	77
C.5. Caso de Uso Histórico Sensores	78
C.6. Caso de Uso Histórico Activación	78
C.7. Caso de Uso Gestionar Zonas	79
C.8. Caso de Uso Gestionar Controladores	80

Capítulo 1

Introducción

En este capítulo se pretende situar en contexto al lector para que disponga de una pequeña base para entender y seguir de forma sencilla el proyecto desarrollado. Para ello se presenta el problema que se quiere solucionar y el estado actual de su posible solución. A mayores se mostrará el contexto y motivación de este Trabajo Fin de Máster y la organización del documento.

1.1. Contexto

El mundo entero se enfrenta a desafíos muy complejos y determinantes en el devenir de la sociedad en lo referente a la gestión de los recursos hídricos. El aumento en el consumo y en la riqueza global de los últimos años ha resultado en una mayor necesidad de alimento. El sector que más porcentaje de uso de agua requiere es el sector agrícola, necesaria para el sistema de regadío. En el caso de España, tenemos la ventaja de contar con una amplia extensión de terreno cultivable y un número de horas de sol que otros países de la Unión Europea no pueden disfrutar.

El principal problema es el agua, esencialmente debido a un régimen irregular de precipitaciones y sequías recurrentes. Sequías y temperaturas extremas más acentuadas con el paso de los años debido, principalmente, al cambio climático que estamos sufriendo actualmente. El análisis de la climatología actual no solo indica temperaturas extremas y sequías más extensas, también provoca tormentas más agresivas que destruyen campos de cultivo y un descontrol total en la periodicidad de precipitaciones.

El uso eficiente del agua en los sistema de riego parece una ámbito de aplicación clave para optimizar el consumo de agua y mejorar la sosteni-

bilidad ambiental, además de para obtener una rentabilidad mayor en la actividad agraria: cualquier agricultor estaría encantado con optimizar el uso del agua y, por lo tanto, gastar una cantidad menor en términos económicos. Como se ha comentado anteriormente, la disponibilidad del agua es limitada; la solución más conveniente y más eficiente pasaría por dar la cantidad de agua que necesita cada cultivo en un momento determinado.

Esto se conoce como **sistemas de riego de precisión**, y son un ámbito muy innovador en los que combinar conocimientos de naturaleza y agricultura con avances tecnológicos. El riego de precisión permite utilizar estos avances tecnológicos en el mundo de la informática para realizar algoritmos de riego óptimos que permitan establecer el momento, la frecuencia y la cantidad de agua adecuados según las características del cultivo, condiciones climatológicas y el estado actual del suelo.

El ámbito de aplicación de estos sistemas es enorme debido a la gran cantidad de aspectos en los que se puede innovar. No solo a nivel de algoritmos automáticos que hagan uso de los modelos de inteligencia artificial más punteros, también existe mejora en el ámbito de la recogida de datos (arquitectura de comunicación IoT, sistemas asíncronos, bajo consumo) o en las propias aplicaciones utilizadas para gestionar y monitorizar toda esta información.

1.2. Motivación

Los conocimientos adquiridos a lo largo de la realización de este Máster han sido muchos y variados, lo que propicia una gran cantidad de diferentes ámbitos en los que enmarcar un trabajo de estas características. La búsqueda y selección de un tema para realizar un Trabajo Fin de Máster es una tarea compleja y que requiere de un análisis en profundidad de varias opciones. El objetivo primordial de un Trabajo de Fin de Máster tiene que ser la puesta en práctica en un espacio más práctico o en un caso de uso real de los conocimientos aprendidos, intentando obtener un resultado que permita complementar a la parte académica. Este afán de innovación y de poner en práctica conocimientos adquiridos parte de una idea inicial que se va dando forma desde su aparición, mejorando con el paso del tiempo hasta obtener la meta planteada.

En el Grado tuve la oportunidad de poder participar en un proyecto de investigación que consistía en desarrollar un sistema de búsqueda basado en términos sobre conjuntos de datos medioambientales, y por lo tanto, poner en práctica varios de los conocimientos adquiridos y aprender nuevos,

como el procesamiento de datos vectoriales y *ráster* y aplicar reglas basadas en lógica difusa. Adicionalmente, en el Máster en Big Data pude aplicar tecnologías y una arquitectura basada en el procesamiento de datos masivos sobre el demostrador previo. Con este Máster mis principios siguen siendo los mismos, poder desarrollar un proyecto final que me permita aplicar nuevas tecnologías y conocimientos adquiridos.

Por esto, en este Trabajo Fin de Máster he querido poner en práctica los conocimientos adquiridos en la mayor parte de las asignaturas y trabajar con equipos hardware como son los microcontroladores, sistemas con los informáticos normalmente no estamos muy acostumbrados a trabajar. La mayor diferencia entre este trabajo y los dos trabajos previos es que se trata de un proyecto personal con un objetivo muy claro de querer ponerlo en funcionamiento para ayudar a resolver la problemática planteada: las pocas alternativas comerciales de sistemas de riego inteligentes que existen en el mercado actual, lo que hace que muchos agricultores locales no puedan disfrutar de las ventajas planteadas por el riego de precisión.

1.3. Problema

El origen que ha surgido como punto de inicio para desarrollar esta idea como desarrollo enmarcado en un Trabajo Fin de Máster era el problema que tenía un familiar del estudiante para controlar la cantidad de agua que suministrarle a una pequeña huerta de fresas debido a la limitación del agua proveniente de un pozo artesanal que se encuentra en la misma huerta. Al tratarse de un problema, de primeras, bastante sencillo, no se creería necesario realizar una inversión elevada en un producto comercial como los que se describirán en el Capítulo 6.

Otro factor muy determinante a la hora de decidir iniciar un desarrollo de este tipo es la gran oportunidad que se tiene de explotar la propia idea en otras huertas de los alrededores con cultivos muy diferentes. Esto implicará una gran cantidad de horas de investigación para mejorar los algoritmos de riego y los sistemas de recogidas de datos, en definitiva, una gran cantidad de trabajo de ingeniería para conseguir un producto mínimo viable que permita solucionar un problema muy importante hoy en día y, con una gran seguridad, que tendrá cada año que pase un mayor impacto en nuestra sociedad: **la gestión eficiente del agua**.

1.4. Organización de la memoria

En esta memoria se documentan las diferentes etapas, fases y desarrollos que se realizaron para poder resolver los objetivos definidos en este TFM. Para ello se explica cada una de las etapas y tareas que se llevaron a cabo así como la justificación de las decisiones tomadas.

Se muestra a continuación una breve descripción de los objetivos de cada uno de los capítulos que forman la memoria:

- En el **Capítulo 2** se muestra el objetivo principal del proyecto y se describen con cierto detalle los objetivos parciales que lo forman para alcanzarlo.
- En el **Capítulo 3** se entra en detalle en los conceptos teóricos necesarios para entender el ámbito del proyecto. En este apartado se entrará en cierto detalle sobre los diferentes elementos hardware utilizados.
- En el **Capítulo 4** se detalla la metodología de desarrollo seguida y se listan, con cierto detalle, el conjunto de herramientas utilizadas para el diseño e implementación del sistema descrito en este documento.
- En el **Capítulo 5** se explicarán algunos de los aspectos más relevantes que se han tenido en cuenta durante el desarrollo del sistema.
- En el **Capítulo 6** se analizan algunos proyectos, sistemas y desarrollos relevantes relacionados con la aplicación de la informática en el ámbito de la agricultura mostrando los resultados obtenidos.
- En el **Capítulo 7** se examina la consecución de los objetivos marcados y se lleva a cabo un análisis de las competencias utilizadas y adquiridas durante el desarrollo de este TFM. Para finalizar este capítulo, se explican las siguientes etapas a seguir en próximas líneas de trabajo que sería interesante abordar así como mejoras a llevar a cabo al proyecto desarrollado.

Para finalizar ester subapartado, se muestra de una manera resumida el contenido de los anexos:

- En el **Apéndice A** se muestran todos los aspectos relativos a la planificación del proyecto.
- En el **Apéndice B** se especifican los requisitos funcionales y no funcionales necesarios para alcanzar los objetivos marcados.

- En el **Apéndice C** se exponen los diferentes aspectos relativos al diseño del sistema. Se han diseñado diferentes diagramas para entender mejor la estructura del sistema y la interoperabilidad entre elementos.
- En el **Apéndice D** se concreta cómo está estructurado y desplegado el sistema desde un punto de vista técnico.
- En el **Apéndice E** se señalan los aspectos más importantes del sistema para que cualquier tipo de usuario sea capaz de utilizar el sistema; en este caso, centrándose en la aplicación web.

Capítulo 2

Objetivos del proyecto

Este capítulo describe el objetivo principal del proyecto y los diferentes objetivos parciales que se necesitan cumplir para alcanzarlo y cumplir con las especificaciones descritas.

Antes de empezar a describir los objetivos del proyecto, es imprescindible comentar que todos los objetivos citados en este capítulo giran en torno al deseo de probar la mayor cantidad de tecnología posible y poner en práctica los máximos conocimientos posibles adquiridos durante la realización del máster.

2.1. Objetivo General

El objetivo principal de este Trabajo Fin de Máster es el diseño e implementación de un **prototipo de sistema de gestión de riego** que permita la recogida, almacenamiento y monitorización de una serie de controladores que permiten la gestión de una serie de sensores que permitan la medición de valores medioambientales como puede ser la temperatura o la humedad, permitiendo la automatización y gestión de un sistema de riego asociado para gestionar el consumo de agua en una huerta de fresas. Este prototipo será la base para futuras mejoras descritas en el Capítulo 7.

Aunque actualmente el sistema permita la gestión de un sistema de riego, la idea es desarrollar el sistema pensando en poder gestionar diferentes sistemas y servicios mediante actuadores genéricos ¹.

Otro aspecto muy importante a comentar es la limitación a la hora de gestionar un único tipo de cultivo, la idea pasaría por poder gestionar un conjunto cada vez mayor de cultivos, lo que implicará unas necesidades de agua claramente diferenciadas: no es lo mismo la cantidad de agua que necesitan unas fresas comparado a una huerta de patatas, ajos o incluso puerros.

A partir de este prototipo, como se ha comentado, la idea a medio y largo plazo es evolucionarlo a un **sistema de gestión automática de cultivos**; no solo gestionar un único cultivo sino gestionar y mejorar el algoritmo de riego para múltiples cultivos y, además, poder gestionar otros aspectos relativos a la agricultura como puede ser la gestión eficiente de un invernadero (apertura de puertas o ventanas, por ejemplo) o la automatización de un sistema de sulfato y/o abono automatizado.

2.2. Objetivos intermedios

Para conseguir cumplir con el objetivo general del proyecto de disponer de un prototipo de recogida y almacenamiento de datos, se necesitarán una serie de objetivos parciales que se detallan a continuación:

- **Diseño y desarrollo de los controladores de sensores:** es imprescindible diseñar y desarrollar un microcontrolador que sea capaz de conectarse con una serie de sensores que permitan obtener diferentes valores medioambientales necesarios para gestionar el sistema de riego, como puede ser la temperatura del aire o la humedad relativa. Este elemento también será el encargado de enviar los datos, siguiendo un estándar compartido por los diferentes módulos, al servidor central.
- **Diseño y desarrollo de los actuadores:** es imprescindible diseñar y desarrollar un microcontrolador que sea capaz de controlar, posiblemente mediante un relé para generalizar lo más posible la solución

¹El concepto de **actuador** se definirá en detalle en el Capítulo 3. De forma resumida, se entiende actuador como un módulo externo al sistema de gestión central encargado de, por ejemplo, una electroválvula que permita regar el cultivo; pero puede también activar otro servicio como un cierre automatizado en un invernadero o la activación de un vallado electrificado

planteada, para este prototipo, el sistema de riego del proyecto. Para ello será necesario diseñar un sistema lo más genérico posible para permitir el control por parte del servidor central de diferentes elementos o servicios, no solo de activar o desactivar una electroválvula.

- **Estructura de almacenamiento de datos de sensores:** estructura que permita almacenar el histórico de valores recibidos de los sensores de cada uno de los controladores que forman el sistema. El sistema de almacenamiento tiene que permitir una flexibilidad a la hora del conjunto de valores almacenados que soporte, por ejemplo, nuevos sensores en el futuro, o la gestión de miles de valores (tanto escalabilidad vertical como horizontal).
- **Diseño y desarrollo de un sistema de recogida de datos:** es necesario contar con un sistema de comunicación entre el servidor (encargado de almacenar los datos) y los controladores (encargados de enviar los valores actuales de los sensores). Es preciso definir un sistema de recogida de datos que siga un estándar de comunicación IoT formado por dos elementos principalmente: un sistema asíncrono de mensajes y un proceso que se encargue de la lectura y almacenamiento de los datos.
- **Diseño y desarrollo de un sistema de procesado:** es obligatorio contar con un servicio encargado encargado de analizar los datos en bruto recibidos de los diferentes controladores para llevar a cabo una gestión eficiente y simple que permita la activación o desactivación del actuador (en este prototipo el actuador será el encargado de activar la electroválvula para permitir el funcionamiento del sistema de riego). Esta gestión será llevada a cabo mediante la implementación de uno o varios algoritmos automáticos que permita analizar los valores de los sensores y determinar si es o no necesario activar el actuador.
- **APIs de gestión:** toda la gestión central será llevada a cabo por un servidor formado por las APIs de comunicación necesarias para gestionar los diferentes elementos del sistema, como los controladores, y la obtención de la información necesaria para visualizar en la web, como los datos del histórico.
- **Interfaz web:** para visualizar el histórico de datos de cada uno de los sensores de los controladores es necesario tener una aplicación web que permita la búsqueda y selección de controladores. El usuario también puede gestionar el sistema de riego (cambiar los modos de

funcionamiento, activar o desactivar el sistema para cada una de las zonas definidas, etc.) y registrar los diferentes elementos que forman parte del sistema, como pueden ser, por ejemplo, los controladores. La aplicación web desarrollada será lo más simple posible, primando la usabilidad frente al diseño y opciones de funcionalidad para obtener una base sólida sobre la que desarrollar funcionalidades futuras.

Capítulo 3

Conceptos teóricos

El objetivo de este capítulo es introducir un conjunto de conceptos teóricos básicos relativos a los microcontroladores y sensores y a los sistemas de riego de precisión, necesarios para entender la solución propuesta encargada de resolver el problema planteado y poder diseñar e implementar el prototipo descrito en el Capítulo 2.

3.1. Riego de precisión

Cualquier cultivo necesita una cierta cantidad de agua para poder subsistir, normalmente obtenida por medio de las lluvias, aunque hay ciertas zonas y épocas del año en la que esta cantidad de agua insuficiente o inexistente. Para solventar este problema existe lo que se conoce como riego de precisión, que permite aplicar con conjunto de técnicas e innovaciones tecnológicas para racionalizar el agua y mantener los cultivos sanos sin desperdiciar nada de agua [1][2].

Evapotranspiración

Una metodología muy utilizada en el ámbito de la ingeniería aplicada a la agricultura para estimar las necesidades de agua de los cultivos es lo que se conoce como evapotranspiración [3][4]. La fórmula utilizada se muestra a continuación:

$$ET_c = ET_0 \times K_c$$

- siendo ET_0 la demanda operativa de la atmósfera o evapotranspiración de referencia y,
- K_c el coeficiente del cultivo que indica el desarrollo de la cubierta del cultivo.

La evapotranspiración de referencia engloba parámetros meteorológicos como temperatura del aire, radiación a nivel del suelo, velocidad del viento o la sequedad del aire, calculándose aplicando la ecuación *Penman-Monteith* [4].

Por lo tanto, se puede apreciar que ET_0 representa un indicador de la demanda climática actual, mientras que K_c varía en función del tipo de cultivo.

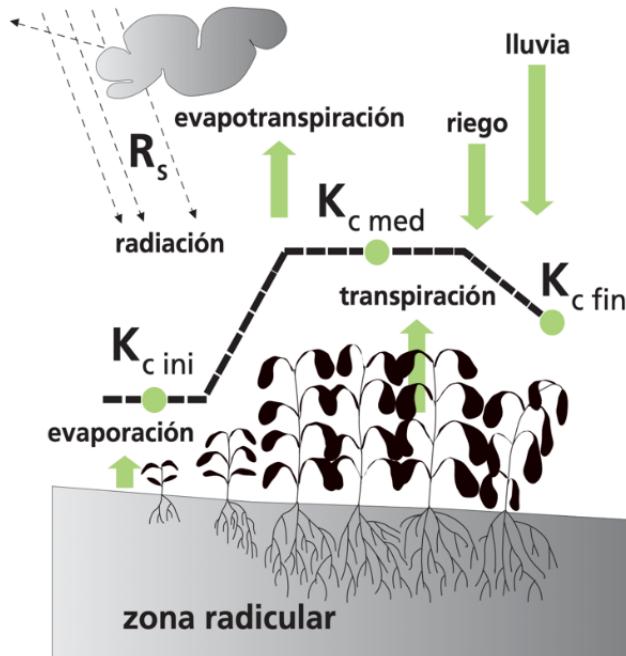


Figura 3.1: Evapotranspiración del cultivo [3]

La Figura 3.1 pretende mostrar un resumen muy sencillo de como influyen los diferentes elementos meteorológicos en la evaporación del suelo y la transpiración de las hojas de los cultivos.

3.2. Bases de datos no relacionales

El uso de las bases de datos no relacionales [5][6], también llamadas NoSQL, están cada vez más en alza y es normal que todos los proyectos hagan uso de al menos un tipo de estas bases de datos debido a que están optimizadas para ser utilizadas con datos que no siguen un modelo determinado y están desarrolladas enfocándose en la escalabilidad, tanto vertical como horizontal.

La principal diferencia con una base de datos convencional es que no se definen tablas, ni esquemas ni relaciones. Los datos se almacenan sin seguir un modelo predefinido, por lo que se puede tener datos con una estructura diferente.

Todas estas características son compatibles con el formato de datos de los históricos que se pretenden almacenar en nuestro sistema.

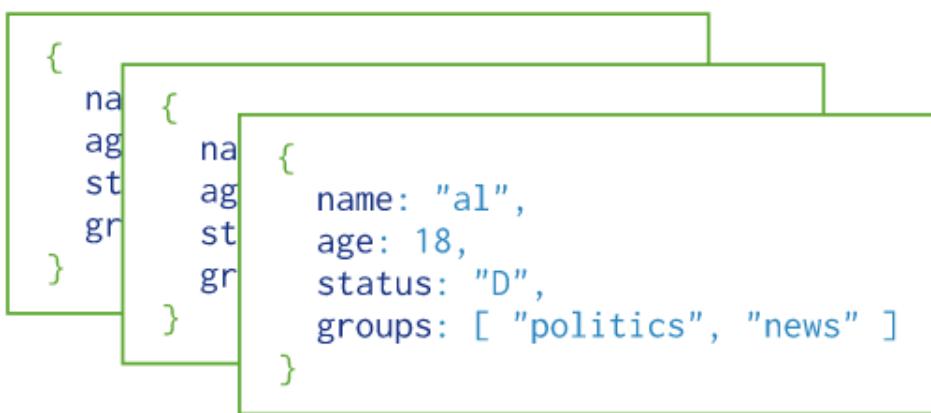


Figura 3.2: Ejemplo de documentos [7]

Se ha decidido el uso de una base de datos basada en documentos, en donde se almacenarán colecciones formados por un conjunto de documentos en formato *JSON*. En la Figura 3.2 se puede apreciar un ejemplo de colección que almacena documentos en formato *JSON* con datos de usuarios.

3.3. Lógica difusa

Para determinar si es necesario o no regar un cultivo determinado, partiendo de la consideración inicial de que todos los cultivos son iguales (que es errónea), a partir de una serie de datos medioambientales obtenidos a través de unos sensores, podemos partir de una serie de condiciones iniciales sacadas del lenguaje natural: *si la humedad es baja se riega, si ha llovido bastante no se riega, si hace muchísimo calor y la humedad es media no se riega.*

Los elementos que acompañan a las propiedades se denominan modificadores y el problema a la hora de codificar esto es que la propia definición de éstos debido a que *alto* o *bastante* tienen connotaciones diferentes dependiendo de la propiedad, e incluso del lugar. Además, *muchísimo calor* puede ser diferente para un agricultor que para una persona que quiera ir a la playa, por ejemplo.

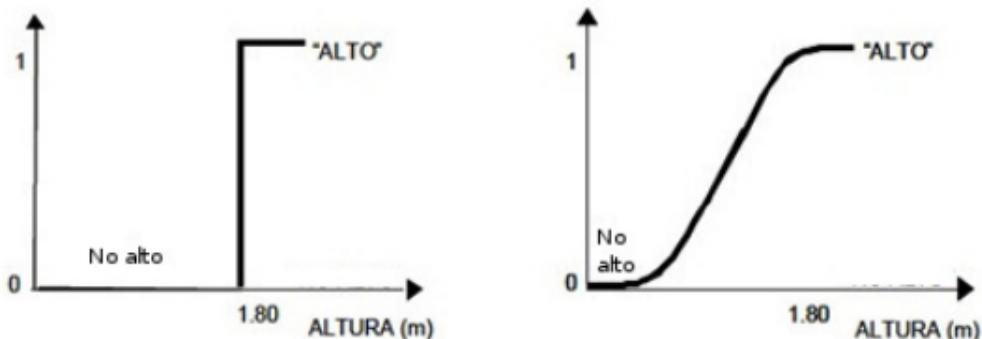


Figura 3.3: Función de pertenencia de *alto* definido mediante conjuntos clásicos y borrosos

Debido a que estos términos resultan imprecisos en su definición es necesario modelados con técnicas de representación del conocimiento como son los *fuzzy sets* [8] que permitan definir restricciones como *temperatura alta* de forma intuitiva para los usuarios. En la Figura 3.3 se puede apreciar de forma general la definición del término *alto* haciendo uso de conjuntos clásicos y borrosos. Si aplicamos este tipo de conjuntos a las propiedades de nuestro sistema, una posible definición (y la utilizada en el algoritmo de ejemplo) se puede visualizar en la Figura 3.4.

Las funciones de pertenencia definidas para las tres propiedades que se analizan para optimizar el sistema de riego no son las más precisas que se podían definir, especialmente la de *lluvia*. Son simplemente un ejemplo, el

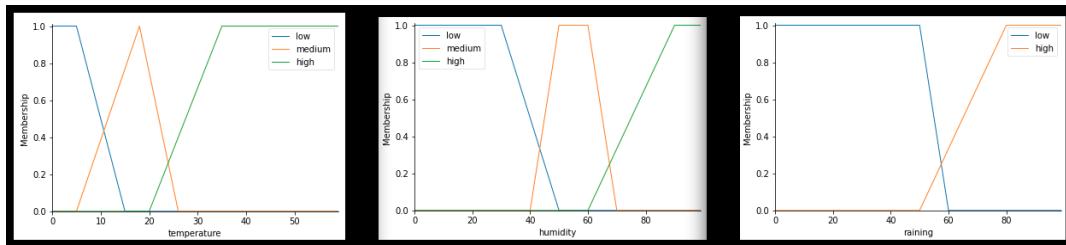


Figura 3.4: Función de pertenencia de *temperatura*, *humedad* y *lluvia*

objetivo del proyecto no es desarrollar un algoritmos de riego súper preciso, para abordar la arquitectura del prototipo.

3.4. Comunicación asíncrona

En sistemas basados en una arquitectura distribuida es muy importante desacoplar el tiempo de ejecución de una tarea hasta esperar una respuesta entre dos módulos independientes. Hacer uso de mensajes asíncronos es una solución muy eficaz para desvincular a los remitentes de los consumidores y evitar el bloqueo del remitente esperando la respuesta del emisor tras finalizar su ejecución. Esta metodología de desarrollo está actualmente muy en alza en arquitecturas distribuidas y basadas en *cloud*.

Patrón Publica-Subscribe

El patrón *publica-subscribe* soluciona el problema de disponer de una cola exclusiva para consumidor mediante los siguientes elementos [9]

- Canal de mensajería de entrada para enviar los mensajes encapsulados por el *publisher*.
- Un mensaje, que se define como un paquete de datos.
- Canal de mensajería de salida para recibir los mensajes por el *subscriber*.
- Un mecanismo, conocido como *broker* para copiar cada mensaje del canal de entrada a los canales de salida correspondientes a los *subscribers*.

Entre las ventajas [9] de este patrón destacar el desacople entre subsistemas comunicados, la escalabilidad y la mejora en la capacidad de respuesta,

mejora la fiabilidad, permite la ejecución en diferido, flexibilidad en la integridad de sistemas basados en diferentes tecnologías y separa el cuello de botella y los puntos de fallo.

Protocolo MQTT

El protocolo utilizado para la comunicación es *MQTT*, un protocolo de comunicación *machine-to-machine* de tipo *message queue* y que se basa en el patrón *Publica-Subscribe* y en la pila TCP/IP como base de su comunicación, lo que lo hace muy versátil, escalable y flexible para la aplicación en entornos IoT [10]. Este protocolo dispone de un mecanismo de calidad de servicio que va desde el nivel 0 (como mucho el mensaje se envía una vez) al nivel 2 (se garantiza la entrega del mensaje).

Se conoce *IoT* (Internet de las Cosas) como el entorno en donde múltiples dispositivos, siendo una inmensa mayoría de ellos dispositivos con múltiples sensores, conectados a Internet se comunican e interaccionan entre ellos para una finalidad común [11].

3.5. Desarrollo de sistemas embebidos

Los sistemas embebidos son sistemas de computación diseñados para realizar funciones muy específica, típicamente aplicaciones de control o sensorización, y cuyos componentes *hardware* están integrados en una placa base. El procesamiento es llevado a cabo por un microcontrolador.

El desarrollo de sistemas embebidos suele ser un proceso tedioso y poco estandarizado, cada microcontrolador tiene métodos muy diferentes de *flashdata* y de compilación, lo que produce un proceso largo y tedioso. Plataformas como *Arduino* y/o *Raspberry Pi* producen un entorno de desarrollo con una metodología muy sencilla que evita a los desarrolladores los problemas a la hora de subir el código.

No se pretende hacer un curso de iniciación a los diferentes tipos y placas de sistemas embebidos que existen, en [12] se explica de una manera muy sencilla y simple los tipos y diferencias que existen entre ellos.

Señales analógicas y digitales

Normalmente este tipo de dispositivos suelen interactuar con dispositivos analógicos y digitales, por lo que es necesario un cierto conocimiento de cómo funcionan los sensores y cómo sus señales son digitalizadas o viceversa

[13]. En la Figura 3.5 se muestran los diferentes tipos de señales con los que suele trabajar en este tipo de ámbito.

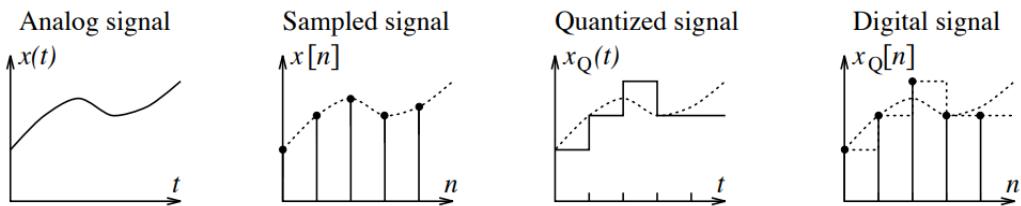


Figura 3.5: Tipos de señales [13]

Para convertir señales, dependiendo del tipo de señal que se parte, tenemos dos tipos de conversores:

- **ADC:** conversor analógico a digital.
- **DAC:** conversor digital a analógico.

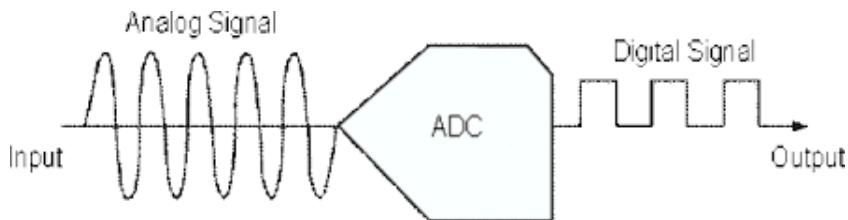


Figura 3.6: Convertidor *analog-to-digital*

Arduino y ESP8266

Las placas de desarrollo escogidas para la lectura de los datos de los sensores y el posterior envío son *Arduino UNO* [14] y el *NodeMCU ESP8266*:

- Se ha escogido una placa basada en *Arduino* debido a la experiencia, muy básica, del desarrollador con este tipo de placas. Esta placa ofrece 14 pines I/O digitales y 6 analógicos, y simplemente se necesitan 5 V para su funcionamiento, lo que la hace una de las placas de desarrollo más utilizada en los trabajos de *IoT*. Este microcontrolador será el encargado de conectarse con los sensores para leer los datos.

- Como la el *Arduino UNO* no tiene módulo de conexión inalámbrica, fue necesario utilizar el *NodeMCU ESP8266* basado en el módulo *ESP8266*. Realmente el controlador podría haberse realizado simplemente con el *NodeMCU*, pero se ha optado por *Arduino* pensando en la limitación de *pines* del *NodeMCU*. Al igual que el *Arduino*, este módulo también es un elemento muy conocido en el mundo *IoT* [15].

Raspberry Pi pico W

Este microcontrolador es una evolución de la placa *Raspberry Pi pico*, clara competidora del *Arduino*. La mejora en esta nueva versión es la integración de un módulo *WiFi* [16]. Debido a que su lanzamiento fue durante el desarrollo del proyecto, se decidió cambiar la idea inicial de utilizar otro *Arduino* para gestionar el *relé* del sistema de riego y probar este nuevo microcontrolador.

Entre las características más importante de esta placa de desarrollo, destacar el uso de *Micropython*¹ como lenguaje de programación por defecto. Cuenta con un modo de bajo consumo y un procesador y memoria con mejores características que la placa de *Arduino*, además de contar con un módulo *Wifi* integrado.

¹Realmente se trata de una *versión* de *Python* para microcontroladores. Más información en <https://micropython.org/>.

Capítulo 4

Técnicas y herramientas

En este capítulo se describirá la metodología de desarrollo utilizada para abordar este TFM y se analizarán diferentes herramientas utilizadas tanto para el diseño como la implementación del proyecto. Se mostrará también un breve análisis entre las alternativas utilizadas en ciertas herramientas así como una justificación de las elecciones llevadas a cabo.

4.1. Metodologías Ágiles

Se ha seguido un desarrollo software basado en metodologías ágiles [17] debido a las ventajas que aporta este tipo de metodología respecto a las metodologías clásicas. El uso de *agile* se está expandiendo muy rápidamente en todos los ámbitos del desarrollo software, porque permite desarrollar rápidamente y gestionar de una manera muy eficiente cualquier cambio que surja a lo largo del ciclo de vida [18]. Dentro de toda las metodologías ágiles se ha decidido utilizar *Scrum*, que se basa en el desarrollo por iteraciones o *sprints* [19].

Se ha decidido utilizar *Scrum* en lugar de una metodología clásica, como cascada, debido a los límites de tiempo marcados y a la necesidad de querer tener entregables funcionales desde el momento inicial para analizar la viabilidad de la solución y poder adquirir un producto mínimo viable lo más completo posible. Además, al tratarse de un proyecto sin ninguna base previa, la incertidumbre y la posibilidad de surgir cambios en la funcionalidad o el enfoque, eran elementos muy críticos a valorar a la hora de elegir el tipo de metodología.

Scrum

Esta metodología se basa en un trabajo interactivo y continuo a lo largo de todo el ciclo de vida, con el principal objetivo de disponer de un entregable al final de cada unas de las iteraciones definidas. Los tres pilares en los que se basa *Scrum* son [19]: transparencia, inspección y adaptación al cambio.

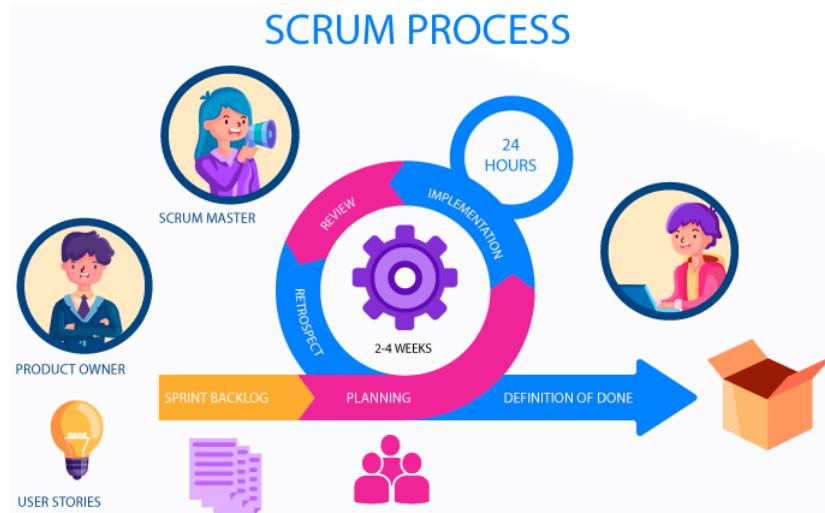


Figura 4.1: Diagrama con el proceso Scrum [20]

La Figura 4.1 muestra los diferentes pasos seguidos en el proceso *Scrum* a la hora de afrontar un *Sprint*. Comentar que este proyecto no se llevaron a cabo los *daily scrum* debido a las limitaciones de disponibilidad y tiempo.

4.2. Despliegue basado en contenedores

Tanto para el desarrollo como el despliegue de la aplicación en producción, se hace uso de una arquitectura basada en contenedores debido a las importantes ventajas a otras soluciones como la virtualización [21]:

- **Modularidad:** Las soluciones basadas en contenedores se enfocan en el despiece de la aplicación en pequeños módulos independientes pero intercomunicados en contenedores diferentes. Esto permite trabajar y actualizar partes de la aplicación sin interferir en el funcionamiento de otra así como aislar las diferentes funcionalidades e identificar mucho más rápido posibles fallos o funcionalidades menos eficientes.

- **Control de versiones:** las imágenes utilizadas para los contenedores pueden cambiar a lo largo del tiempo, por lo que la gestión de versiones es un aspecto primordial a la hora de controlar los contenedores.
- **Restauración:** Gracias al uso del control de versiones de las diferentes imágenes que utiliza nuestro sistema para desplegar los contenedores, es muy sencillo gestionar la compatibilidad entre versiones así como la restauración a versiones antiguas. Restaurar una versión anterior es simplemente apagar el contenedor actual y desplegar un nuevo contenedor indicando la versión de la imagen.

En este caso se va a utilizar *Docker*[22][23] como la tecnología encargada de la definición de las imágenes de los contenedores y el despliegue de los mismos. La principal ventaja de esta solución respecto a una alternativa basada en máquinas virtuales es la eliminación de gestionar el hardware del dispositivo así como toda la administración de sistemas y recursos. De todas maneras, el uso de *Docker* no provoca estrictamente el dejar de utilizar *máquinas virtuales*; muchas soluciones desplegadas en el ámbito empresarial se basan en la combinación de ambas tecnologías.

Para gestionar múltiples contenedores de una manera rápida y sencilla se va a utilizar una herramienta por línea de comandos llamada *docker-compose* [24].

4.3. Sistemas empotrados

Una parte muy importante de este proyecto se ha basado en el diseño, desarrollo y despliegue de una serie de servicios enmarcados en el ámbito de los sistemas empotrados: **servicio de recogida y envío de datos de los sensores y servicio de gestión de activación del relé** del sistema de riego. Como medio de desarrollo y despliegue se han utilizado microcontroladores, que se pueden definir como *chips* integrados capaces de ejecutar las órdenes grabadas en su memoria [25]. En el Capítulo 3, se hace una breve introducción a nivel teórico de su funcionamiento para tener una cierta base.

Las placas de desarrollo se muestran a continuación:

- *Arduino UNO* y *NodeMCU ESP8266* para el controlador de los sensores.

- *Raspberry Pi pico W* para el actuador encargado de la gestión del *relé* del sistema de riego.

Sensores

No se pretende hacer una descripción técnica en profundidad de todos estos elementos hardware porque realmente no es el objetivo del trabajo, pero se encuentra importante citar los sensores utilizados: TODO:

- **Sensor FC-28:** sensor de humedad que permite medir la humedad del suelo por la variación de su conductividad, por lo que la precisión que se obtiene en el valor no es una medición absoluta, pero para un sistema de riego es más que suficiente [26]. La ventaja de este sensor, aparte de ser el más utilizado en este tipos de sistemas, es que permite obtener la medición como valor analógico y digital. En este proyecto, como se quiere un valor relativo, la opción escogida es utilizar el modo digital, que devuelve valores entre 0 (agua) a 1023 (aire).
- **Sensor FC-37:** sensor de lluvia que permite detectar la presencia de lluvia midiendo la variación de conductividad, siguiendo un esquema similar el *FC-28* [27]. La medición puede ser por valor analógico o digital cuando se supera un cierto umbral regulado a través de un potenciómetro integrado. En este caso, simplemente interesa saber si está lloviendo porque este sensor no permite calcular la cantidad de agua acumulada, por lo que se hará uso del valor analógico que será *LOW* en caso de no detectar agua y *HIGH* en presencia de lluvia.
- **Sensor LM-35:** sensor que dispone de un circuito de control integrado que proporciona una salida de voltaje proporcional a la temperatura (10mV por cada grado centígrado), a diferencia de otras soluciones que miden la temperatura a través de una resistencia eléctrica [28]. La precisión a temperatura ambiente es de 0.5 °C y el rango de temperaturas se encuentra entre los -55 °C y los 150 °C, lo que lo hace un sensor bastante eficiente y barato.
- **Relé 5V** para controlar la activación de la electroválvula. En [29] se puede encontrar más información respecto al funcionamiento y esquema de montaje.

4.4. Microservicios

Se ha seguido una filosofía de desarrollo basada en servicios y microservicios a la hora de abordar el desarrollo de la parte lógica del sistema que conforma el *backend*. Las opciones en *Python*, principalmente para el desarrollo de *backend*, están entre *Flask* y *Django*. Debido a la filosofía minimalista, sencillez a la hora de desplegar un servicio y a un mejor rendimiento, se ha escogido *Flask*.

Flask es un *microframework* desarrollado en *Python* para el desarrollo de aplicaciones web basadas en el patrón *MVC*. En este proyecto, se va a utilizar simplemente para el desarrollo de microservicios, pero puede ser utilizado como *framework full-stack*.

Destaca su versatilidad y un desarrollo basado en extensiones bajo demanda: si necesitas una funcionalidad nueva, seguramente haya una extensión que la cubra. Esta es una de las principales diferencias con *Django*; *Django* ofrece un *framework* completo de funcionalidades desde el inicio, se utilicen o no se utilicen.

4.5. Otras herramientas

En esta sección se listarán otras herramientas utilizadas que no se considera necesario entrar demasiado en detalle:

- **Arduino IDE:** Editor de código fuente para la codificación, compilación y carga del programa en arduino con las librerías de *ESP8266WiFi* y *PubSubClient* cargadas.
- **VS Code:** Editor de código abierto que permite el soporte de múltiples lenguajes. En este proyecto se ha utilizado para programar en *Python*, *Vue*, *HTML*, *JavaScript*, *CSS* y *Docker*.
- **Python 3:** Lenguaje de programación de código abierto que destaca por su fácil aprendizaje, sintaxis muy sencillas y un abanico de módulos diferentes para ámbitos muy dispares.
- **paho-mqtt:** librería *MQTT* para *Python* para llevar a cabo la comunicación asíncrona con el *broker*.
- **scikit-fuzzy:** librería para la definición de las reglas basadas en lógica difusa en *python* para el algoritmo de activación del actuador.

- **unittest**: librería para *Python* que permite la automatización de tests.
- **Vue**: Framework progresivo JavaScript para el desarrollo de aplicaciones web que está ganando cada vez más popularidad. Es considerado uno de los 3 reyes actualmente en el desarrollo web junto con *React* y *Angular*.
- **Vuetify**: *Framework* de diseño de componentes para interfaces de usuario para *Vue* basado en el estándar de *Material by Google*.
- **MariaDB**: *Fork* del Sistema de Gestión de Bases de Datos *MySQL* utilizada para gestionar la base de datos relacional.
- **MongoDB**: Sistema de bases de datos orientado a documentos *JSON* y de código abierto, utilizado para almacenar los históricos de datos del sistema.
- **MongoDB Compass**: Herramienta interactiva de código abierto para ejecutar *queries*, optimizar y analizar los datos almacenados en una instancia *MongoDB*.
- **Postman**: herramienta, que nació como una extensión de *Chrome*, que permite realizar peticiones de una manera simple para testear APIs de diferentes tipos. De forma adicional, en muchas ocasiones se utiliza como gestor documental para especificar las APIs de una aplicación.
- **GitHub**: Plataforma *cloud* para el control de versiones (repositorios creados con *git*) y la integración y despliegue continuo utilizada para la gestión de cambios del software.
- **Overleaf**: Plataforma *cloud* para la edición colaborativa de proyectos en *L^AT_EX*, utilizada para la escritura de la memoria.

Capítulo 5

Aspectos relevantes del desarrollo del proyecto

En este capítulo se van a describir los aspectos que se han considerado más importantes a la hora de abordar el desarrollo software de este proyecto durante todo el ciclo de vida del mismo. Se citarán los problemas con un mayor impacto que se tuvieron que afrontar así como las decisiones que se tomaron y sus implicaciones, especialmente aquellas que no son triviales.

5.1. Inicio del proyecto

La idea del proyecto surgió del gran impacto que puede tener aplicar la informática al ámbito de la agricultura y cómo este tipo de soluciones puede resultar ser un nicho con gran impacto en términos de sostenibilidad y gestión eficiente del agua en los cultivos.

Además, se encontró una aplicación inmediata en un entorno real que podría servir como base para desarrollar un servicio bajo demanda con mayores funcionalidades y una arquitectura sencilla, fácil de usar e integrar con otras soluciones comerciales debido a la estandarización de las comunicaciones (solo se necesita desarrollar un *middleware* para la comunicación entre elementos internos y externos).

5.2. Formación

El proyecto requiere de una serie de conocimientos técnicos que no suelen ser introducidos en el Grado, tales como el uso de un sistema de

colas para permitir una comunicación asíncrona en un entorno *IoT*, el diseño y desarrollo de sistemas embebidos o el uso de *frameworks JavaScript* para el desarrollo de aplicaciones web. A la hora de llevar a cabo este proyecto, el desarrollador ya contaba con experiencia previa desarrollando servicios utilizando microcontroladores en entorno *IoT* y experiencia en varios proyectos/aplicaciones a nivel industrial en entornos distribuidos y con comunicación asíncrona basada en sistemas de colas como *Celery* o *Kafka*.

Cabe destacar la importancia de la comunidad [StackOverflow](#) para resolver dudas a nivel de programación, tanto en el ámbito *frontend* como *backend* y [al blog](#) de Luis Llamas para temas relativos al uso de sensores en sistemas embebidos.

5.3. Arquitectura

Una vez se han establecido los objetivos de este proyecto, el segundo paso ha sido diseñar la arquitectura del sistema identificando los módulos más importantes que lo van a formar. Para ello se han tenido en cuenta una serie de consideraciones iniciales:

- Es necesario que los diferentes sistemas embebidos que vayan a formar parte del sistema tengan que *hablar* un idioma común entendible por el resto de módulos.
- Todos los servicios que formen parte del servidor central serán desplegados en contenedores *Docker*.
- Los módulos identificados serán lo más independiente posibles.
- Se dispone de una red inalámbrica de conexión para la comunicación entre módulos.
- Las funcionalidades de los sistemas embebidos será lo más simple y limitada posible para obtener elementos muy sencillos de mantener a lo largo del tiempo.
- Se tienen que hacer uso de microservicios (desacoplar *frontend* y *backend*).

En la Figura 5.1 se puede apreciar la arquitectura diseñada.

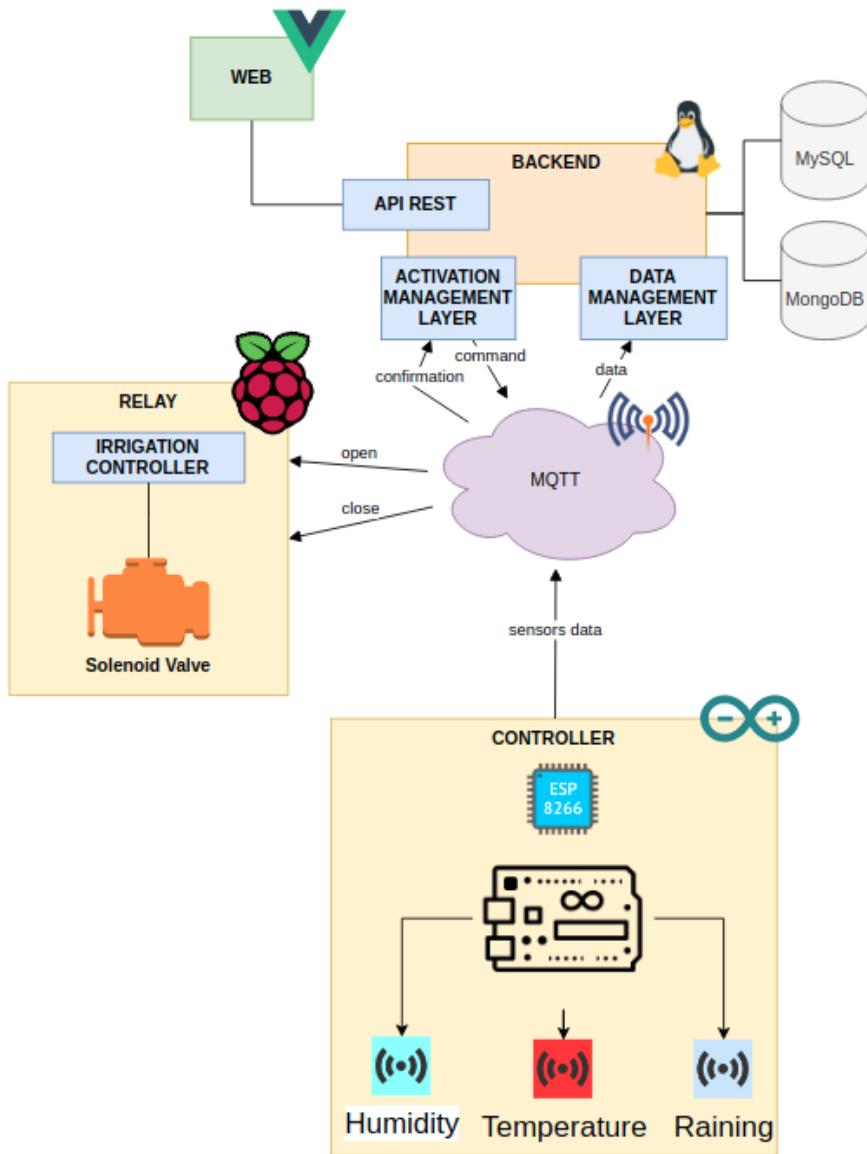


Figura 5.1: Arquitectura del sistema

Se explica a continuación algunas consideraciones de los módulos identificados:

- Debido a que los módulos tenían que ser lo más independiente posibles, se subdivide el sistema en 4 grandes grupos: los microcontroladores, el

servidor central con la mayor parte de la lógica del sistema, las bases de datos y la aplicación web.

- Para mantener al mínimo las dependencias entre los módulos del servidor central, se ha utilizado un desarrollo basado en microservicios, lo que permite hacer módulos más *testeables* y flexibles a cambios.
- Se hace uso de dos bases de datos diferentes para almacenar datos de tipo diferente de una manera más eficiente. Aunque sería factible almacenar los datos de sensores y de activación en una base de datos MySQL, la flexibilidad y la eficiencia se verían afectadas.
- Para finalizar, la comunicación se lleva a cabo mediante un protocolo ligero basado en un broker publica-subscribe permitiendo un desarrollo basado en cola de tareas en los diferentes microservicios del servidor central. Gracias al uso de esta comunicación asíncrona, desacoplamos la ejecución de cada módulo y evitamos esperar a que otro módulo termine su ejecución para obtener un valor, por ejemplo.

En el Apéndice C se muestra con cierto detalle los diferentes pasos seguidos a la hora de diseñar el sistema.

5.4. Diseño de las bases de datos

Para modelar las diferentes tablas, y colecciones, de las bases de datos se han tenido en cuenta una serie de aspectos.

- En el caso de la base de datos relacional:
 - Se ha decidido agrupar los controladores de una misma zona física o ámbito en zonas diferentes.
 - Para cada conjunto de controladores (a.k.a. zonas), se asigna un tipo en función a la acción llevada a cabo por el actuador. Para el prototipo se consideran simplemente **zonas de monitorización** (básicamente se reciben datos de los sensores y no se hace nada adicional a la visualización del histórico) y **zonas de riego** que permiten activar un sistema de riego a partir de los datos de sensores.
 - En principio, no se quieren guardar los datos de los actuadores para desacoplarlos de la aplicación principal.

- Se registran usuarios con sus credenciales (usuario y contraseña) para limitar el acceso a la aplicación web mediante el uso de *tokens* de acceso (posibilidad futura de utilizar el *token* para acceder a las propias APIs).
- Los diferentes roles de los usuarios serán **administrador** y **usuario invitado** sin privilegios.
- En el caso de la base de datos no relacional:
 - Se crearán dos colecciones diferentes para el histórico de datos de sensores y de activación del actuador.
 - La colección de datos de sensores almacenará los valores de cada propiedad medioambiental y las unidades de medida (por ahora esto no se utiliza) y la información del controlador y zona al que pertenece.
 - En el caso del histórico de activación, simplemente se quieren guardar los intervalos, por lo que con indicar el área y un rango temporal, sería suficiente.

5.5. Servicios y Microservicios

Como se ha comentado anteriormente, una de las consideraciones indicadas era la independencia de los diferentes módulos de los servicios centrales. Para ello se han separado los servicios del sistema en dos aplicaciones diferentes:

- Un servicio de usuarios y *tokens* de acceso (servicio de autenticación).
- Y un servicio de gestión encargado de almacenar y utilizar los datos de los controladores.

El principal problema que se consideró al plantear esta solución era que pasaba con los 'procesos' encargado de comunicarse con los microcontroladores. Después de un análisis en profundidad de proyectos con el mismo tipo de arquitectura y basándose en la experiencia del desarrollador se llegaba a la conclusión de que había 2 posibles soluciones factibles de aplicar ante este escenario:

- Lanzar todos los procesos y levantar la Api REST en un mismo proceso utilizando hilos.

- Aplicar un desarrollo basado en microservicios y separar e independizar la ejecución de esos 'procesos'.

La solución escogida, obviamente, fue la segunda debido a que era la que mejor se adaptaba a la arquitectura y consideraciones planteadas.

Servicios de gestión y autenticación

Para la comunicación entre la aplicación web y las bases de datos, tanto de usuarios, de gestión como los históricos de datos de la base de datos no relacional, se ha decidido utilizar Api REST, posiblemente el *estándar de facto* en la comunicación *frontend* y *backend*. Los *endpoints* definidos utilizan métodos *GET* para la obtención de información, *POST* para crear un nuevo recurso, *PUT* para actualizar información de un recurso y *DELETE* para la eliminación de un recurso.

Debido a la experiencia del desarrollador con *Flask* y *Python*, se ha optado por utilizar este *microframework*.

Los patrones de diseño utilizados han sido los siguientes:

- Se ha utilizado una combinación del patrón **Repository** y **Data Access Layer** para abstraer el desarrollador del acceso al sistema de gestión de datos, en este caso, las bases de datos.
- Se ha utilizado el patrón **Model-View-Presenter** para la gestión de peticiones en las Api REST. Este modelo permite separar la representación lógica de los datos de la representación de la vista (*response*), siendo el *presenter* el que simula el comportamiento de un controlador (patrón *MVC*).
- Se ha utilizado el patrón **Singleton** para impedir tener múltiples instancias de las conexiones a la base de datos o de la propia instancia de *Flask*.
- Se ha utilizado el patrón **Observer** para gestionar el modelo de comunicación basado en Publica-Subscribe.
- Por último, para gestionar los diferentes algoritmos que se pueden aplicar por parte del microservicio que gestiona la activación del actuador, se ha utilizado el patrón **Strategy**.

Microservicios

De los dos microservicios desarrollador, el encargado de procesar los datos del histórico de datos de sensores para saber si activar o no el actuador es el algoritmo más complejo, aunque también es bastante sencillo de entender.

- Se recorren las zonas que sean de tipo riego.
- Se obtienen los últimos valores de los sensores de cada controlador de una zona.
- Se obtienen el porcentaje del tiempo que ha llovido en las últimas 24 horas.
- Se aplica un algoritmo determinado para, a partir de los datos medioambientales, saber si activar o no el actuador.
- Se emite el comando al actuador.

En cuanto al microservicio encargado de almacenar datos, simplemente hacer eso: recibe el mensaje por parte del controlador, comprueba que la zona y controlador están registrados y los almacena en la base de datos.

5.6. Comunicación asíncrona

Como se descrito en el subapartado anterior, el intercambio de información entre aplicación web y *backend* se hace utilizando Api REST. Esto provoca que la aplicación web esté muy acoplada a los servicios que utilizad del servidor central; si no hay *backend* la web no puede ni siquiera arrancar, lo resulta ser lo más lógico. Para no tener este problema con los microcontroladores, la comunicación entre éstos y los microservicios del servidor central se decidió utilizar una comunicación asíncrona que permite este desacople; además, esta tecnología está basada en un desarrollo por colas de tareas, para no tener que esperar una respuesta del lado servidor.

El protocolo de comunicación utilizado es *MQTT*, cuya arquitectura de comunicación se puede apreciar en la Figura 5.2, que destaca por su uso en entornos *IoT* con un ancho de banda muy limitado debido a que los mensajes de intercambio suelen ser bastante ligeros. Por experiencia propia del desarrollador, mensajes de más de 1MB de contenido suelen dar algún que otro problema a la hora de la recepción. Para gestionar todo esto se ha utilizado *Mosquitto*, *broker Open Source* implementado en Java que destaca por ser liviano y escalar muy bien.

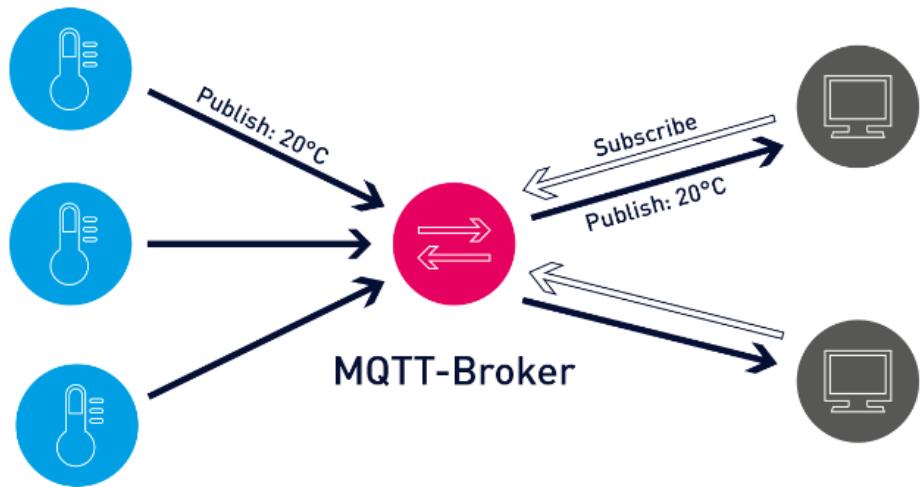


Figura 5.2: Arquitectura de comunicación

La comunicación asíncrona tiene 3 tipos de mensajes:

- Mensaje de tipo *dato* para el envío de **datos de sensores** enviado por el microcontrolador; al enviar los datos del histórico, que serán almacenados en una MongoDB, el contenido del mensaje estará en formato *JSON* para simplificar el procesado y guardado.
- Mensaje de tipo *comando* para **activar/desactivar el actuador** enviado por el servidor central.
- Mensaje de tipo *comando* para **confirmar la activación/desactivación del actuador** enviado por el servidor central.

5.7. Aplicación Web

Durante el desarrollo de la aplicación web, los únicos problemas efectivos que se encontraron fueron:

- Decidir si utilizar *Vue 2* o *Vue 3* como *framework* de desarrollo.
- Decidir si utilizar alguna librería de componentes de interfaz estilo *Bootstrap* o diseñar y codificar todo desde cero a base de *CSS* y *JavaScript* puro.

El segundo dilema analizado tuvo una muy fácil solución, debido a los tiempos que se manejaban y al gran trabajo qué supone desarrollar todos los componentes desde cero, se decidió que sería necesario utilizar una librería de componentes. El principal problema fue si la versión de *Vue* debido a que la versión 3 todavía no está en un estado estable y las librerías que soportan la nueva versión son escasas.

Aún con todo esto, se decidió utilizar *Vue 3* debido a la gran cantidad de mejoras que ofrece y que, a finales de este año, se prevé su despliegue definitivo. En cuanto a la librería utilizada, se elige *Vuetify* debido a la experiencia del desarrollador y al soporte de *Vue 3*.

Para el diseño de la aplicación se tuvo en cuenta el minimalismo y el hecho de contar con el menor número de vistas posibles para reducir código a mantener y flexibilizar posibles cambios futuros. Otro aspecto que ayuda en estas características es el hecho de que la aplicación web desarrollada se trata de una SPA [30], lo que permite que todas las vistas de la aplicación se concentren en una única página web sin tener que hacer *reloads* por parte del navegador.

Debido a que la aplicación web desarrollada es la parte *visible* y central del proyecto, se describen en los siguientes apartados algunos aspectos destacables de la misma.

Diseño

En este subapartado se destacarán los aspectos más importantes del diseño de la aplicación web, más información relativa al diseño del proyecto se puede encontrar en el Apéndice C. La Figura 5.3 muestra el mapa web de la aplicación, con las diferentes *vistas* a las que puede acceder el usuario (destacadas en color verde), así como las diferentes acciones y gestiones que se pueden llevar a cabo en la vista principal (destacadas en color naranja).

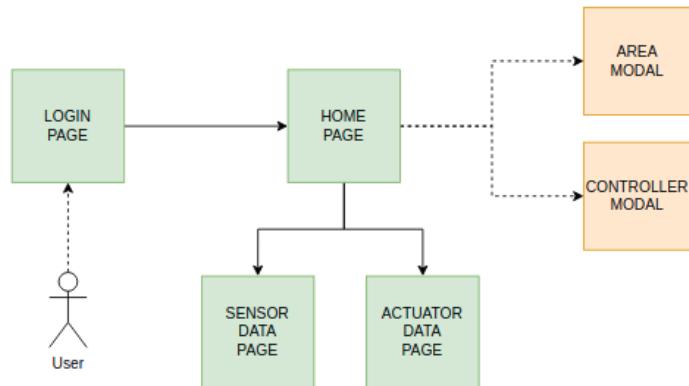


Figura 5.3: Mapa Web de la Aplicación

Tal y como se puede apreciar en la Figura 5.3, el usuario tiene que pasar obligatoriamente por la vista de *login* para acceder a la aplicación. La vista principal de la aplicación está formada por el listado de zonas y los controladores que las forman, así como los botones de gestión; tal y como se muestran en la Figura 5.4.

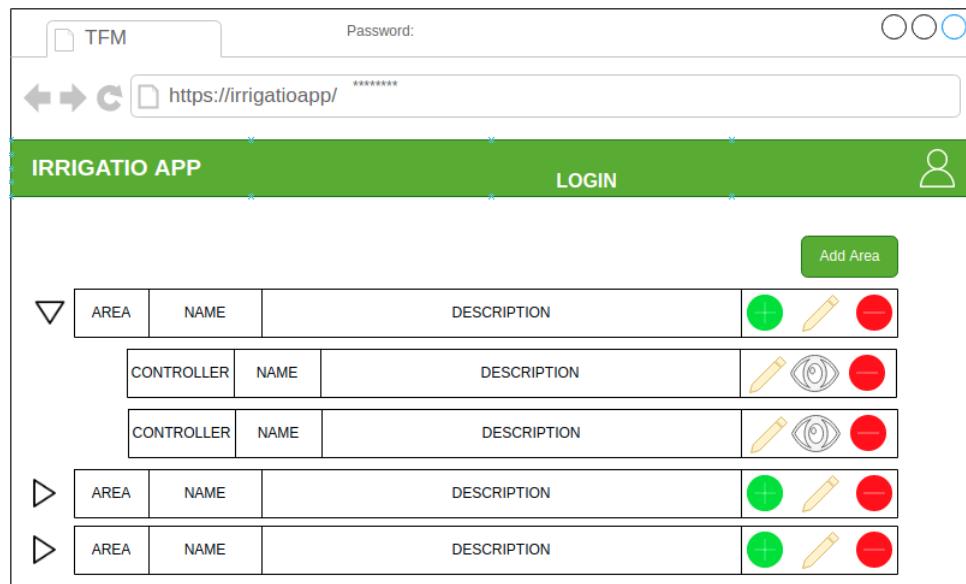


Figura 5.4: Boceto: Página de Inicio

En la vista principal, el usuario puede gestionar tanto las zonas como los controladores mediante botones bien definidos a través de modales, lo que permite aprovechar la misma vista inicial y seguir viendo el listado

de elementos en el *background*. Desde la vista inicial, el usuario también puede acceder a las vistas de histórico, en donde puede seleccionar un determinado rango temporal y visualizar la información relativa a los datos de sensores (temperatura, humedad y valor de lluvia) así como a los períodos de activación del relé que gestiona el riego para cada una de las zonas. La Figura 5.5 muestra un boceto de estas vistas.

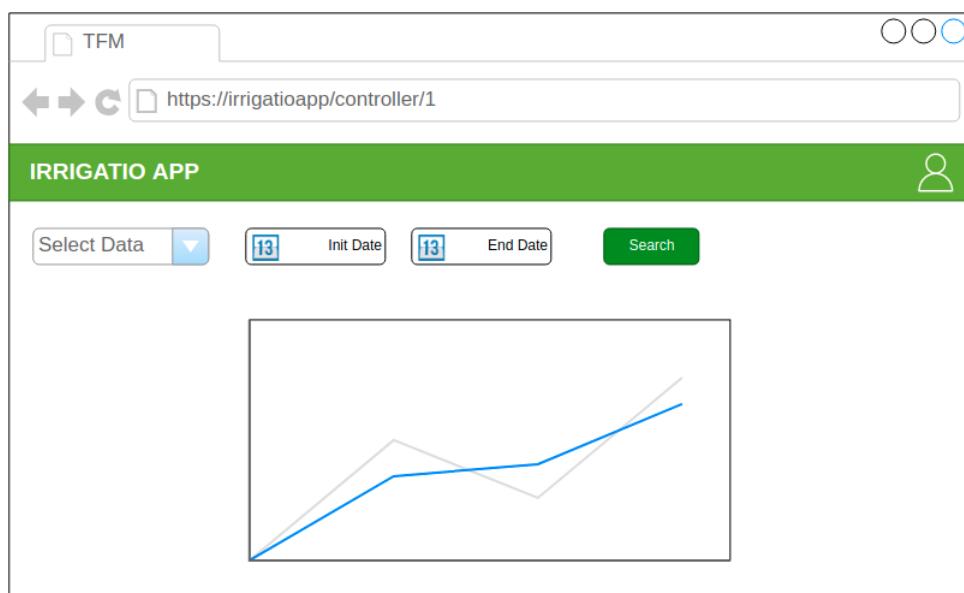


Figura 5.5: Boceto: Visualización de histórico de datos

Comunicación

En cuanto a la comunicación con el sistema, se han desarrollado una serie de servicios diferenciados basados en *API Rest*:

- Gestión de *tokens* de autenticación para el control de acceso a la aplicación.
- Gestión de zonas.
- Gestión de controladores.
- Gestión de los históricos de datos de sensores y actuadores.

Implementación

La Figura 5.6 muestra el aspecto final de la vista de inicio de la aplicación web, con el listado de las zonas y controladores y los botones de gestión.

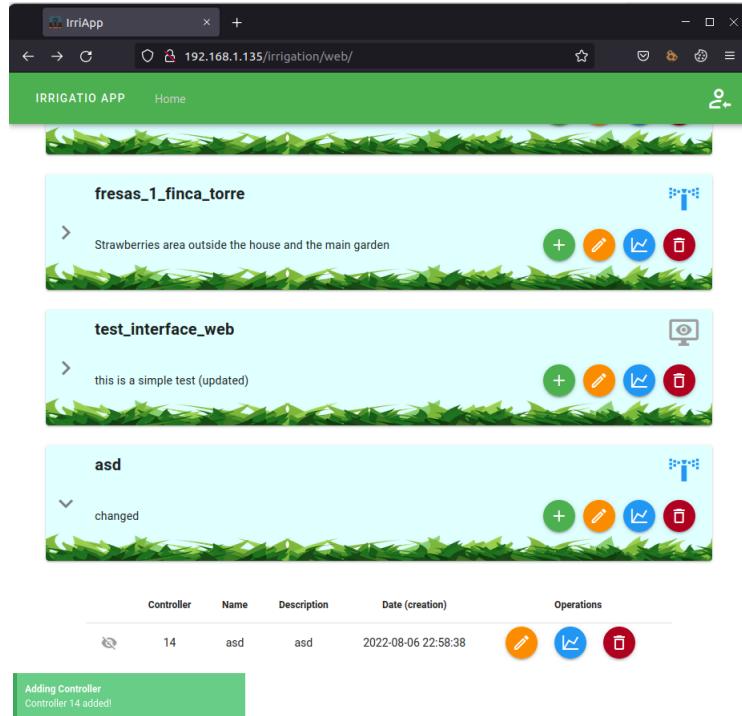


Figura 5.6: Aplicación web: Página de Inicio

Un ejemplo de modal para gestionar nuevas áreas se puede apreciar en la Figura 5.7, en donde el usuario solo necesita especificar un nombre identificativo de zona, una descripción y un tipo. Todos los modales desarrollados siguen un mismo patrón de minimalismo y sencillez para facilitar su uso al usuario final. Adicionalmente, se ha primado el hecho de disponer de una aplicación web sencilla sobre la que desarrollar futuras funcionalidades.

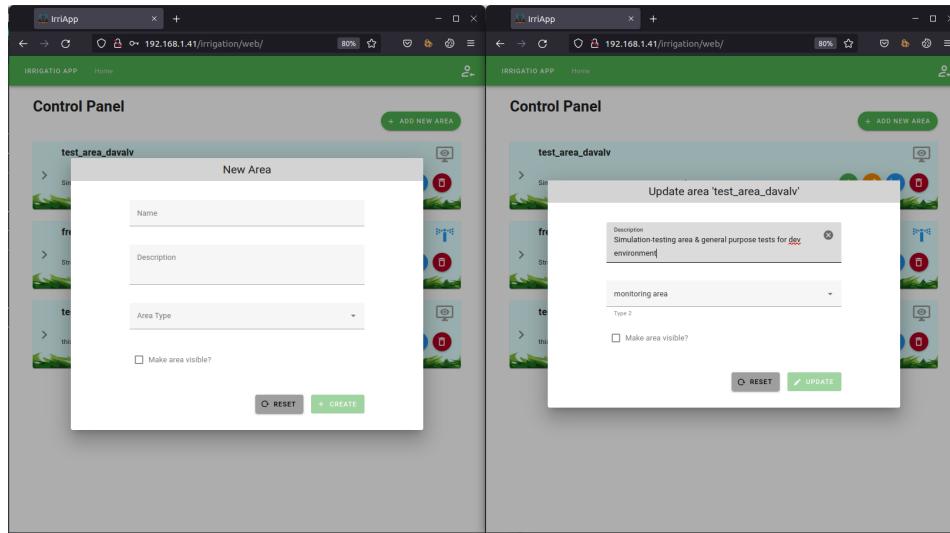


Figura 5.7: Aplicación web: Gestión de Zonas

Por último, en la Figura 5.8 se puede apreciar un ejemplo de intervalo temporal con datos de los sensores (valores de los sensores obtenidos por medio del simulador del controlador).

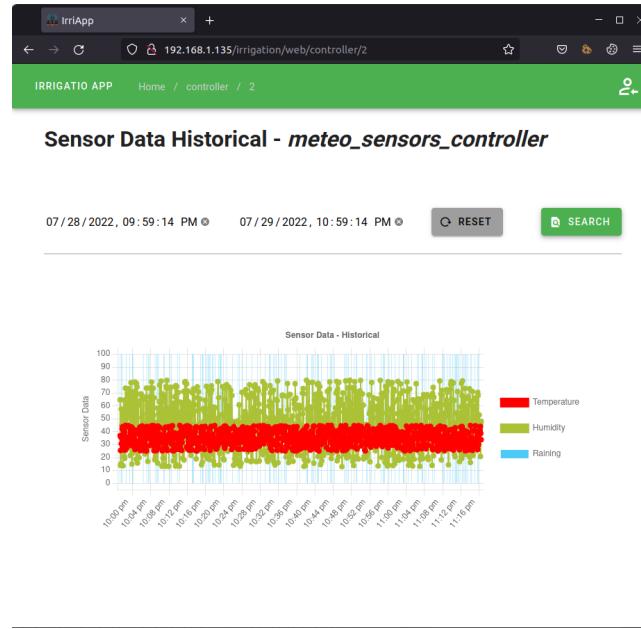


Figura 5.8: Aplicación web: Monitorización de sensores

Una descripción más en profundidad de cómo utilizar la aplicación y más ejemplos de capturas de pantalla de las diferentes vistas, pueden ser encontradas en el Apéndice E. En el Apéndice D también se pueden encontrar capturas con los diferentes casos de uso de fallo de la aplicación.

5.8. Testing

La metodología seguida a la hora de desarrollar y ejecutar un plan de pruebas ha sido diferente para cada uno de los módulos que forman el sistema.

- En el caso de los sistemas embebidos, debido a la poca experiencia en proyectos reales por parte del desarrollador, se definieron las diferentes pruebas que se querían realizar en papel y se ejecutaron a mano. Para ello se apoyó en el uso de los terminales series, en el caso del controlador, y la metodología basada en *print* para los actuadores.
- En el caso de los servicios de gestión, tanto Apis REST como los microservicios, se definieron los planes de pruebas utilizando la librería de *python unittest*, que permite programar test automáticos, lo que facilita mucho la ejecución de test cada vez que se cambia algo en el código o se añade una nueva funcionalidad. En el caso del servicio de autenticación, por falta de tiempo, para probar el funcionamiento de las Apis REST se hizo uso de *Postman* y una terminal con acceso a la base de datos para comprobar que cumplía el plan de pruebas.
- Para finalizar, en el caso de la aplicación web, al tratarse de una aplicación muy sencilla, no se consideró necesario utilizar ninguna librería de *Vue* para programar los test. Las pruebas se hicieron con un navegador web y aplicando los pasos definidos en el plan de pruebas.

5.9. Despliegue en producción

Debido a las limitaciones de tiempo y al alcance del proyecto, no se ha podido llegar a esta etapa del ciclo de vida y desplegar el sistema en una huerta de fresas de prueba para analizar y comprobar que todo funciona correctamente, además de poder analizar, optimizar y mejorar el algoritmo encargado de la gestión de activación del actuador en zonas de riego.

Otros aspectos que impidieron llegar a esta etapa se muestran a continuación:

- Diseño y fabricación de una carcasa para guardar los microcontroladores y no exponerlos a la intemperie.
- Retrasos en la recepción logística de los materiales hardware.

Los pasos necesarios para desplegar el sistema, tanto en un entorno de desarrollo como de producción, se describen en el Apéndice D. Debido a que se ha llevado a cabo un desarrollo modular y, gracias al uso de contenedores para el despliegue de los servicios, el proceso de puesta en producción es muy sencilla:

- Despliegue de los servicios (contenedores *Docker*) en el servidor central.
- Compilar y cargar el programa en el controlador.
- Compilar y cargar el programa en el actuador.

Capítulo 6

Trabajos relacionados

El objetivo del proyecto era el diseño, desarrollo y despliegue de un sistema de riego automatizado que permitiese almacenar los valores de los sensores en una base de datos y disponer de un servicio o módulo que permitiese procesar y analizar esos datos y calcular si es o no es necesario activar el sistema de riego. Una vez desarrollado este prototipo la idea era una evolución previsible a un sistema más genérico con mayores funcionalidades como las comentadas en el Capítulo 2.

Estamos ante un ámbito de gran interés, por lo que obviamente lo desarrollado y planteado en este documento no es una idea única e innovadora; en el contexto de que ya existen empresas aplicando estas pautas y premisas para desarrollar productos finales y funcionales que permite una gestión más óptima del sistema de riego. Adicionalmente, se considera imprescindible destacar la gran cantidad de proyectos y contribuciones *Open Source* que se pueden encontrar en plataformas como *Github* o *Gitlab* así como diferentes *kits* que se pueden encontrar en *Amazon* o *Aliexpress* para desplegar tu pequeño sistema de riego automatizado.

Este capítulo presentará muy brevemente algunas de estos productos así como un par de publicaciones que permiten aplicar el uso de algoritmos de inteligencia artificial para la gestión eficiente del agua.

6.1. Gardena

Gardena[31] no es un sistema de riego automático como tal, se define como un sistema de cuidado de tu jardín a través del uso de diferentes productos sensorizados y una plataforma *Cloud* que permite al usuario la

visualización y monitorización y gestión de los diferentes elementos que se encuentre en el jardín.

Las Figuras 6.1 y 6.2 muestra una serie de productos que forman parte de la familia de productos de Gardena, entre los que se pueden encontrar un cortacésped automático, un actuador muy parecido al desarrollador en este documento encargado de activar y desactivar el flujo de agua (se supone que de un sistema de riego por dispersión), un sensor encargado de la captación de información del estado del jardín como puede ser la temperatura o la humedad del suelo o un sistema de control encargado de la gestión por zonas y sensores.



Figura 6.1: Productos sensorizados de Gardena [31]



Figura 6.2: Productos sensorizados de Gardena (II) [31]

El sistema es muy interesante para personas que suelen ir de viaje y quieran tener un control casi absoluto del estado de su jardín, pero como se dice, solo funciona para jardines. En este caso se trata de un sistema aplicado a un ámbito de servicio o *hobby*, no es un sistema pensado para un ámbito agrícola. De todas formas, el sistema de comunicación de los sensores y la posibilidad de una aplicación móvil son ideas a tener en cuenta.

6.2. Libelium

Libelium[32] es una empresa tecnológica fundada en Zaragoza como una *spin-off* de la Universidad de Zaragoza dedicada principalmente al

diseño y fabricación de hardware y *SDKs* (kits completos de desarrollo de software) para redes de sensores inalámbricos integrados en sistemas complejos basados en tecnología *IoT* para proyectos en el ámbito de las *smart cities* y la sostenibilidad medioambiental. Destaca principalmente por el gran abanico de ámbitos que abarca, por una tecnología muy madura y unos tiempos mínimos de comercialización.

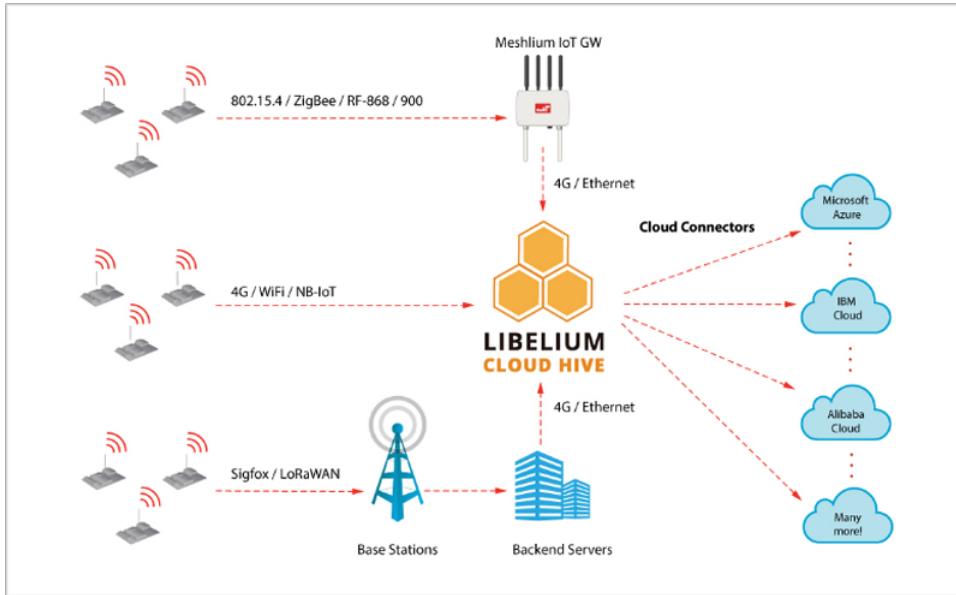


Figura 6.3: Arquitectura Libelium Cloud [32]

No hay mucha información relativa a la arquitectura utilizada ni a los elementos que forman su gama de soluciones basados en la sostenibilidad y el correcto uso del agua en la agricultura, pero parecen contar con un sistema *Cloud* centralizado, denominado *Libelium Cloud*, que permite la gestión y recogida de datos de sensores y aplicar una serie de algoritmos basados en las necesidades de agua según el cultivo controlado. La Figura 6.3 muestra de forma muy esquemática dicha arquitectura.

6.3. WheaterTRAK

WheaterTRAK[33] es una de las soluciones de sistemas de gestión de riego desarrollado y comercializado por la empresa estadounidense HydroPoint. La empresa proporciona, como la mayoría de ellas, tanto los sistemas de control de los sistemas de riego (Figura 6.5) como los sensores y las electroválvulas automatizadas (Figura 6.4).



Figura 6.4: Dispositivos de riego de WheaterTRAK [33]

Las electroválvulas encargadas de activar y desactivar el flujo de agua cuentan con un microcontrolador integrado que se comunica con el sistema central mediante comunicación directa.



Figura 6.5: Controladores de riego de WheaterTRAK [33]

6.4. Sativum

La última plataforma del listado de sistemas y plataformas relativas a la gestión de datos de sensores y a la sostenibilidad de los sistemas de riego es Sativum[34], una plataforma del Instituto Tecnológico Agrario de Castilla y León que pretende centralizar y estandarizar la forma de visualizar y gestionar los datos de sensores de diferentes parcelas agrícolas.

Es una plataforma que sirve de modelo para el desarrollo de este proyecto debido a que esta plataforma permite unificar en un solo sistema la gestión y visualizar de datos muy diferentes provenientes de diferentes fuentes de datos:

- Acceso a datos sobre el suelo, el clima y el cultivo de parcelas mediante el acceso a una red de sensores, base de datos e imágenes de satélites.
- Ayuda en la toma de decisiones sobre el cultivo como gestionar la fertilización o la cantidad de agua que es necesario aportar al cultivo.

En la Figura 6.6 se puede apreciar una de las múltiples vistas de Sativum para permitir llevar a cabo la monitorización de los datos recolectados de la red de sensores.

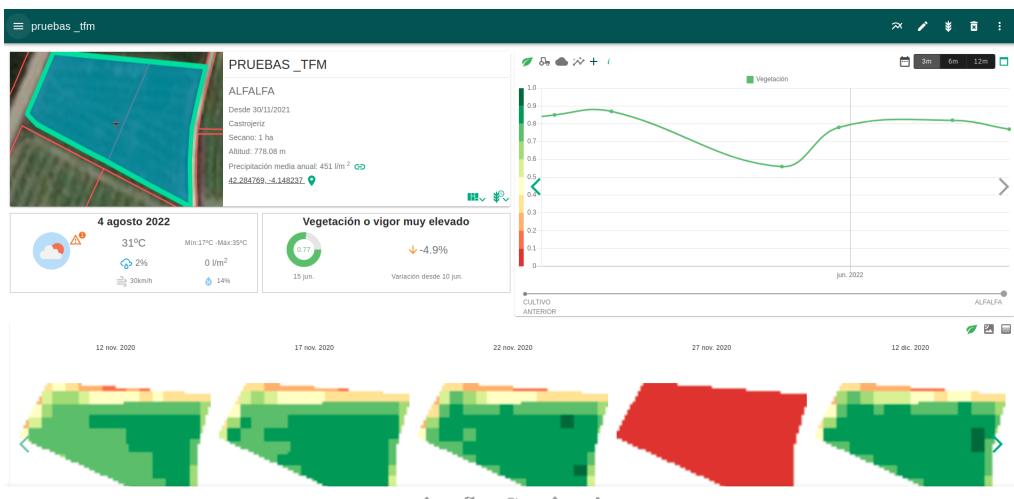


Figura 6.6: Aplicación Web de Sativum [34]

6.5. Aplicando modelos basados en Inteligencia Artificial

Para finalizar este apartado, aunque realmente no forma parte de los objetivos de este trabajo, existen muchas aplicaciones de técnicas de inteligencia artificial basadas en *fuzzy logic* o *neural networks* para optimizar los algoritmos de riego, vamos a destacar 2. En ambos casos se tratan de algoritmos genéricos, no se hace un análisis en profundidad de las necesidades de agua por tipo de cultivo.

En [35] se propone un sistema de decisión basado en lógica difusa sobre la temperatura y la humedad para obtener el grado de apertura del sistema de riego para enviar más o menos agua. Los resultados obtenidos parecen

bastante prometedores: con mucha humedad el sistema de riego está al mínimo de flujo mientras que con unos valores bajos activa el máximo flujo.

Por último, en [36] se propone un sistema de activación basado en una red neuronal artificial (ANN) dentro de una arquitectura distribuida basada en un servidor central desplegado en el *cloud*. La red neuronal toma como entrada los valores de temperatura, humedad y cantidad de agua recogida de lluvia para determinar el grado de necesidad de agua del cultivo.

Como resumen de este capítulo es importante destacar que existen muchas empresas a nivel internacional que se dedican a la instalación de sistemas de riego automático, según describen en sus páginas web, pero no describen casi ningún aspecto relativo a la arquitectura de su sistema. Son empresas que se dedican a aplicar conocimientos de ingeniería para solucionar diferentes problemas e instalar un sistema en una localización física (como el caso de *WheaterTRAK*). Otras empresas que ofrecen este tipo de servicio de instalación de un sistema de riego son Novagric¹ o Llaberiagroup², ambas empresas dedicadas principalmente a proyectos de ingeniería civil que dan solución a problemas de muchos ámbitos distintos.

La solución propuesta en este documento trata de diseñar una arquitectura *Cloud 'centralizada'* en la que poder conectar diferentes sensores sin necesidad de una instalación compleja, definiendo una serie de mensajes de comunicación para que los diferentes módulos se entiendan.

¹Más información en <https://www.novagric.com/es/riego/sistemas-de-riego/riego-automatico>.

²Más información en <https://www.llaberiagroup.com/proyectos/>.

Capítulo 7

Conclusiones y Líneas de trabajo futuras

En este capítulo se analizarán los objetivos conseguidos y se describirá brevemente el listado de tareas llevado a cabo para completarlos. También se mostrarán posibles líneas de trabajo futuras y aspectos a mejorar del sistema desarrollado que se han encontrado a lo largo del desarrollo.

7.1. Objetivos cumplidos

El principal objetivo de este trabajo era el diseño y desarrollo de un **prototipo** de sistema que permitiese la recogida y almacenamiento de datos medioambientales para su posterior uso en la gestión de la activación de un sistema de riego complementario. De forma complementaria, este prototipo tiene que proporcionar una aplicación de usuario que permita monitorizar los datos obtenidos de los sensores y permitir gestionar los diferentes controladores que forman las diferentes zonas registradas.

A continuación se muestra un listado de las tareas llevadas a cabo para completar los objetivos parciales planteados en el Capítulo 2:

- Selección de las tecnologías para los diferentes microcontroladores que forman el sistema.
- Selección de un sistema de comunicación asíncrona para la interoperabilidad entre los diferentes módulos que forman el sistema: microcontroladores y servidor central.

- Diseño de una estructura de almacenamiento para la información recibida por parte de los controladores.
- Diseño de una estructura de almacenamiento para la información relativa a las zonas que forman el sistema y a los controladores de cada una de las zonas.
- Diseño del diagrama de conexiones electrónicas (sensores) del controlador.
- Diseño del diagrama de conexiones electrónicas (*relé*) del actuador.
- Desarrollo del código fuente que permite la recogida de datos de los sensores por parte del *Arduino UNO* y el posterior envío al servidor mediante mensajes *MQTT* por parte del *NodeMCU ESP8266*.
- Se realizan pruebas de funcionamiento del controlador: comunicación *Arduino UNO-NodeMCU ESP8266* y comunicación *NodeMCU ESP8266-MQTT*.
- Desarrollo del código fuente del actuador que permite la activación/-desactivación del relé.
- Se realizan pruebas de funcionamiento del actuador: comunicación *MQTT-actuador* para activar/desactivar el *relé*.
- Diseño, desarrollo y despliegue de un servicio de recogida de datos de los controladores para su posterior almacenamiento en la base de datos *MongoDB*.
- Se realizan pruebas de funcionamiento simulando el envío de datos y comprobando que se insertan correctamente en la base de datos.
- Diseño, desarrollo y despliegue de un servicio de procesamiento que permite la gestión de los actuadores.
- Se realizan pruebas de funcionamiento simulando el envío de los comandos de gestión de la activación mediante eventos *MQTT*.
- Diseño, desarrollo y despliegue de un servicio central con las Api REST del sistema encargadas de gestionar las zonas y controladores y obtener el histórico de datos, tanto de los sensores de los controladores como de la activación de los actuadores.

- Diseño, desarrollo y despliegue de una aplicación web que permite al usuario visualizar y gestionar las diferentes zonas y los controladores que forman cada área y monitorizar los datos de los históricos.
- Se han realizado pruebas de interfaz para comprobar el correcto funcionamiento de la web (en caso de ausencia de datos) y la usabilidad de la web.
- Despliegue de los diferentes módulos que forman el sistema mediante el uso de contenedores *Docker*.
- Se ha validado el sistema completo.

7.2. Competencias adquiridas

El principal objetivo de este máster, y por tanto, de este Trabajo Fin de Máster, es adquirir una serie de competencias que permitan el alumno mejorar como Ingeniero Informático [37]. La finalización de este trabajo permitan demostrar el buen uso de las competencias adquiridas por el alumno a lo largo del curso académico. A continuación se destacan algunas de ellas:

- Resolver problemas en entornos nuevos o poco conocidos dentro de casos de uso más amplios, multidisciplinares y complejos.
- Integrar nuevas tecnologías y sistemas propios de la Ingeniería Informática en entornos más amplios y complejos.
- Diseñar, evaluar y desplegar aplicaciones y sistemas basados en computación distribuida y asíncrona.
- Modelar, diseñar, definir la arquitectura, desplegar, gestionar, operar y mantener aplicaciones, redes de comunicación, sistemas, servicios y contenidos informáticos.
- Crear y explotar sistemas en entornos virtuales y *dockerizados*.
- Diseñar y desarrollar aplicaciones y servicios informáticos en sistemas empotrados y ubicuos.
- Aplicar métodos matemáticos, estadísticos y de inteligencia artificial para modelar, diseñar y desarrollar, sistemas y servicios inteligentes.

- Analizar, diseñar y desarrollar sistemas funcionales que permiten la solución de un supuesto real combinando técnicas de ingeniería de software e inteligencia artificial.
- Aplicar conocimientos avanzados acerca de las técnicas y herramientas empleadas en la implementación de los aspectos de presentación e interacción del usuario con el entorno web.
- Proyectar, calcular y diseñar productos, procesos e instalaciones en todos los ámbitos de la ingeniería informática.
- Planificar y gestionar el tiempo de desarrollo a lo largo del ciclo de vida del proyecto.
- Obtener una sensibilidad hacia temas medioambientales.

7.3. Problemas y limitaciones del sistema

Aunque el resultado obtenido cumplió con los objetivos marcados, se encontraron algunos problemas y/o limitaciones a lo largo del desarrollo que se pueden modificar para futuras versiones del sistema de gestión de riego (en un futuro se hablará de *sistema de gestión automático de cultivos*):

- Las zonas registradas solo pueden tener un tipo asociado, lo que limita bastante, pensando de cara al futuro, la funcionalidad de los actuadores.
- Las zonas solo pueden disponer de un actuador que gestiona el sistema de riego. Sería necesario modificar el servicio de procesamiento para gestionar diferentes tipos de zonas de una manera más eficiente. Es importante tener en cuenta que el actuador desarrollado es un sistema muy genérico que simplemente gestiona la activación de un *relé*, por lo que no va a ser necesario cambiar nada.
- Debido a las limitaciones de tiempo y al alcance del proyecto, los algoritmos que gestionan los actuadores son bastante sencillos.
- Imposibilidad de descargar los datos de los históricos en formato *CSV* o *JSON*.
- El sistema se ha desplegado en una *Raspberry Pi 4*, por lo que existen problemas de escalabilidad futuras.

- Los datos medioambientales son bastante limitados, sería interesante llevar a cabo un análisis en profundidad de posibles nuevos sensores a añadir.
- El protocolo de comunicación utilizado entre los módulos se basa todo en *HTTP*, lo que disminuye la seguridad del sistema de manera considerable.

En el siguiente apartado se comentan algunas líneas de trabajo que se consideran seguir para mejorar el sistema, considerando de que se parte de aplicación sólida y correcta.

7.4. Mejoras y Trabajo futuro

A partir del análisis de los problemas identificados a lo largo de todo el desarrollo y plasmados en el apartado anterior, se han planteado y propuesto un conjunto diverso de mejoras y ampliaciones a implementar de cara al futuro para mejorar el sistema propuesto y poder ponerlo en producción.

La arquitectura diseñada permite disponer de una muy buena y completa base para la construcción de un sistema de gestión automática de cultivos más avanzado. El sistema desarrollado ha permitido construir unas bases muy sólidas para las futuras mejoras que se planean llevar a cabo.

Se diferencia en este apartado una primera etapa cuyo objetivo es el despliegue en un entorno real de producción, y una segunda etapa que consistirá en aplicar mejoras de mayor entidad sobre las funcionalidades desarrolladas en este prototipo.

En cuanto a los pasos a seguir para conseguir desplegarlo en producción son los siguientes:

- Diseñar la placa *PCB* del controlador para la conexión de los sensores y el microcontrolador.
- Diseñar la placa *PCB* del actuador para conectar el *relé* y el microcontrolador.
- Enviar a fabricar los modelos 3D de las carcasa de los microcontroladores.
- Analizar la periodicidad del envío de datos y de la gestión de los actuadores.

- Estudio de campo de los diferentes algoritmos desarrollados para decidir qué cuál aporta mejores soluciones (estudio previo a ponerlo en producción, una vez desplegado todo será necesario llevar a cabo otro análisis más profundo).

Se plantean diversas mejoras que permitirán dotar al sistema de una mejor funcionalidad y una mayor eficiencia:

- Cambiar los *Arduino UNO* de los controladores por *Raspberry Pi Pico W*.
- Gestionar la activación y desactivación de más de un actuador por zona.
- Permitir que cada una de las zonas definidas permitan tener asociado más de un tipo para permitir generalizar la funcionalidad de los actuadores que forman la zona.
- Generalizar el uso del actuador para permitir gestionar la activación de otros elementos mecánicos, como puede ser la gestión de un invernadero inteligente.
- Diseñar y desarrollar una vista en la interfaz web que permita visualizar en una vista el resumen de todos los controladores por zona.
- Diseñar y desarrollar una vista en la interfaz web que permita la gestión de nuevos usuarios.
- Analizar, diseñar, desarrollar y desplegar nuevos tipos de actuadores que permitan generalizar el uso del sistema de gestión.
- Mejorar el algoritmo de gestión de activación para las zonas de riego utilizando un modelado basado en valores de necesidad de agua como puede ser la *evapotranspiración del cultivo* [1][2].
- Utilizar un protocolo de comunicación seguro como *HTTPs*, lo que implica generar una serie de certificados a utilizar en los diferentes módulos y generar algún tipo de servicio que permita la actualización del certificado de forma automática.

Apéndice A

Plan de Proyecto Software

Para la realización de proyectos software, tanto de grandes como de pequeñas dimensiones, es necesario utilizar una gestión correcta y eficiente a lo largo de todo lo que dure el ciclo de vida del proyecto. De esta forma se reduce la posibilidad de aparición de riesgos y de errores. El proceso de gestión consiste en aplicar ciertas técnicas y habilidades con el principal objetivo de iniciar, desarrollar, supervisar y controlar todo el proyecto a lo largo de su ciclo de vida.

A.1. Introducción

Esta sección describe la organización llevada a cabo para cumplir con los objetivos marcados para completar este TFM, desde la idea original del proyecto, pasando por el análisis y desarrollo de los diferentes módulos que forman el sistema, hasta la documentación y preparación de la presentación y la demostración del sistema.

A.2. Planificación temporal

El objetivo de este apartado es mostrar los diferentes *sprints* que se han realizado para obtener el prototipo del sistema automático de riego.

Sprint 0: Inicio del proyecto

Los objetivos de este *sprint* inicial se detallan a continuación:

- Planificar la organización del TFM.
- Analizar y elegir una metodología de trabajo acorde a los objetivos marcados.

Debido a la naturaleza del proyecto y al tiempo limitado de desarrollo y alcance del proyecto, se ha acordado que la mejor metodología de trabajo que se adapta a este proyecto es *Scrum*. Para ello, como resultado de este *sprint* se define el número de *sprints* de los que constará este proyecto así como la duración de cada uno de ellos:

- El proyecto estará formado por un total de **17 sprints** sin contar este inicial.
- Cada *sprint* tendrá una duración aproximada de **1 semana de trabajo**, considerando un único desarrollador a tiempo completo, con un único día de descanso por semana.

Sprint 1: Estudio del estado del arte

Los objetivos de este *sprint* se detallan a continuación:

- Creación del proyecto en una plataforma *git* que permita el seguimiento.
- Creación la plantilla de la documentación en L^AT_EX.
- Buscar proyectos y documentación que apliquen un enfoque parecido al definido.
- Analizar alternativas y estado del arte del riego de precisión.

La plataforma escogida para la gestión del proyecto software es *GitHub*, mientras que para la documentación se ha escogido *Overleaf* debido a la posibilidad de trabajo colaborativo entre tutores y alumno, a la facilidad de uso y ausencia de instalación.

Relativo al estado del arte, se han encontrado los típicos *kits* de riego automatizados basado en horas fijas y utilizados para los jardines, pero sin ningún control sobre valores de sensores externos ni una aplicación real en un entorno agrícola.

Sprint 2: Diseño e interfaces de comunicación

Los objetivos de este *sprint* se detallan a continuación:

- Diseñar la arquitectura del sistema identificando los diferentes módulos que forman el sistema.
- Especificar los diferentes casos de uso.
- Analizar posibles tecnologías *IoT* para llevar a cabo una comunicación asíncrona basada en mensajes.
- Definir los mensajes asíncronos utilizados por los diferentes módulos que componen el sistema.
- Analizar lenguajes de programación que se van a utilizar en los diferentes módulos.

Se obtiene un diseño inicial de la arquitectura del sistema identificando los módulos y partes más diferenciables del sistema:

- Interfaz Web.
- Servidor central:
 - Apis REST.
 - Servicio de recolección de datos.
 - Servicio de gestión y comunicación con los actuadores.
- Microcontroladores:
 - Controlador encargado de la recogida y envío de datos.
 - Actuador encargado de controlar la activación del relé

Finalmente se han analizado diferentes tecnologías de comunicación asíncrona como *Kafka*, *RabbitMQ* y *MQTT* y se ha escogido *MQTT* por su sencillez y facilidad de uso, además de ser la alternativa más liviana.

Sprint 3: Captación de datos de sensorización

Los objetivos de este *sprint* se detallan a continuación:

- Analizar placas de desarrollo y posibles sensores a utilizar.
- Diseñar conexiones eléctricas.
- Estudiar el funcionamiento de los sensores (analógico o digital).
- Desarrollar el código del controlador para la recogida y envío de datos.

Se ha escogido la placa de desarrollo *Arduino UNO* para la captación de datos a través de los sensores de temperatura, humedad y lluvia. Debido a que la placa de desarrollo no cuenta con un módulo de conexión a red, fue necesario utilizar el módulo *ESP8266* para llevar a cabo la conexión y envío de datos. De esta forma separamos la funcionalidad de recogida de la del envío, lo que puede facilitar en un futuro cambiar la placa de desarrollo por otro modelo.

Sprint 4: Procesamiento y almacenamiento de datos

Los objetivos de este *sprint* se detallan a continuación:

- Diseño de la base de datos relacional para almacenar información relativa a la aplicación, como pueden ser los controladores (modelo Entidad-Relación y relacional).
- Diseño de la base de datos no relacional para almacenar los históricos de datos de los sensores para cada controlador (modelado de los documentos).
- Analizar gestores de bases de datos.
- Creación de un microservicio encargado de recolectar datos a través de MQTT y almacenarlos en la base de datos no relacional.
- Diseño y ejecución de pruebas de unitarias y de integración.
- Creación de un contenedor *Docker* para el despliegue.

Se diseñan los modelos de datos de ambas bases de datos y se escogen utilizar *MariaDB* como base de datos relacionado y *MongoDB* como base

de datos no relacional. El uso de *MongoDB* es debido a que la información de los sensores va a ser recibida por el servicio de recolección en formato *JSON*.

Dentro de la base de datos relacional se deciden gestionar los controladores por zonas para poder tener un mayor control sobre los diferentes elementos.

Sprint 5: Integración I: Controlador - Servicio de recolección

Los objetivos de este *sprint* se detallan a continuación:

- Diseño y ejecución del plan de pruebas:
 - Comprobar que los datos enviados por parte del controlador son recibidos por el servicio de recolección y almacenados en la base de datos no relacional.
 - Comprobar que gestiona la existencia de los controladores.
- Comprobar despliegue conjunto.

Al tratarse de un prototipo, la comunicación entre los diferentes módulos se hace mediante un protocolo *HTTP*. Esta decisión ya ha sido comentada como uno de los supuestos del proyecto.

Sprint 6: Gestión servidor y APIs de comunicación

Los objetivos de este *sprint* se detallan a continuación:

- Analizar posibles *frameworks* a utilizar en el lado servidor para el diseño de las Apis REST.
- Diseñar servicios de acceso a datos aplicando patrones de diseño.
- Diseño y ejecución de pruebas de unitarias y de integración.
- Creación de un contenedor *Docker* para el despliegue.

Debido a la sencillez y a la experiencia de uso por parte del desarrollador, se ha escogido *Flask* como framework de desarrollo de los servicios de comunicación del servidor del sistema.

Sprint 7: Interfaz de usuario para la gestión y visualización de datos

Los objetivos de este *sprint* se detallan a continuación:

- Diseño y modelado de la aplicación web mediante *mockups*.
- Analizar *frameworks JavaScript* para la creación de la aplicación web.
- Creación e inicialización de la aplicación web.
 - Vista inicial para mostrar el conjunto de zonas y controladores.
 - Vista para la gestión de zonas.
 - Vista para la gestión de controladores.
 - Vista de monitorización para visualizar el histórico de valores de los sensores.
- Creación de un contenedor *Docker* para el despliegue.

Los *mockups* se han diseñado utilizando *Draw.io* mediante el uso de elementos muy básicos para entender la finalidad de cada una de las vistas de la aplicación. En cuanto al *frameworks JavaScript*, se ha escogido *Vue* debido a la gran popularidad que está ganando en estos años y a la experiencia con él por parte del desarrollador.

Sprint 8: Integración II: Apis REST - Aplicación Web

Los objetivos de este *sprint* se detallan a continuación:

- Diseño y ejecución del plan de pruebas (se parte de datos almacenamos previamente en las bases de datos):
 - Comprobar la información de las zonas listadas en la vista inicial.
 - Comprobar los controladores correspondientes a cada una de las zonas.
 - Comprobar la información de los controladores.
 - Comprobar la vista de gestión de zonas: crear una nueva zona, modificar información de la zona y eliminar zona.
 - Comprobar la vista de gestión de controladores: crear un nuevo controlador, modificar controlador y eliminar controlador.

- Comprobar que los datos enviados por parte del controlador y almacenados en la base de datos no relacional por parte del servicio de recolección se visualizan en la vista de monitorización.
- Comprobar despliegue conjunto de aplicación web y servidor.

Sprint 9: Programar el actuador de la electroválvula

Los objetivos de este *sprint* se detallan a continuación:

- Analizar viabilidad de utilizar una nueva placa de desarrollo.
- Estudiar el funcionamiento del relé.
- Diseñar conexiones eléctricas.
- Analizar si utilizar eventos asíncronos o Apis REST para la comunicación actuador-servidor.
- Desarrollar el código del controlador gestionar los eventos de gestión y activar o desactivar el relé.
- Utilizar eventos asíncronos para notificar de la activación o desactivación del relé.

Una semana antes de empezar este *sprint* salió al mercado el nuevo microcontrolador diseñado por *Raspberry Pi* que contaba con un módulo de conexión inalámbrica. Se decidió analizar la viabilidad de utilizar este nuevo microcontrolador debido a la principal ventaja de contar con un módulo de conexión, así como utilizar *MicroPython* como lenguaje de desarrollo.

Debido a las limitaciones de tiempo y al entorno de desarrollo, se decide utilizar un *led* como sistema de simulación de la electroválvula de riego. Tanto el sistema de activación como las conexiones electrónicas son las mismas: el sistema permite pasar corriente sobre el relé, en este caso de ejemplo se iluminará la luz, en el caso real, la electroválvula se activaría y dejaría pasar el agua.

Para finalizar este *sprint*, fue necesario analizar la viabilidad a nivel técnico y de seguridad de utilizar una comunicación síncrona o asíncrona. Debido a la ventajas que aporta, se ha optado por seguir utilizando el sistema de eventos asíncronos.

Nota importante: debido a problemas de tiempo en la logística de envío, la recepción del microcontrolador se retrasó, por lo que se analizó

la viabilidad de probar esta nueva placa de desarrollo o hacer toda la programación utilizando Arduino. Debido al margen de tiempo y a las tareas pendientes, se decidió posponer los *sprints* 10 y 12 y saltar directamente al 13.

Sprint 10: Gestión del riego

Los objetivos de este *sprint* se detallan a continuación:

- Diseño de los algoritmos de las zonas de riego para determinar si es necesario regar.
- Análisis de algoritmos que hacen uso de *fuzzy logic* para determinar si es necesario regar o no.
- Creación de un microservicio encargado acceder a los datos del histórico de cada uno de los controladores de las zonas de riego y emitir los comandos al actuador mediante eventos asíncronos.
- Diseño y ejecución de pruebas de unitarias y de integración.

Nota: debido al rápido desarrollo y puesta en marcha del actuador, se añadieron una serie de objetivos:

- Almacenamiento de los períodos de activación en la base de datos no relacional (histórico que indica cuando estuvo activado el relé).
- Diseño y desarrollo de una API REST para obtener el histórico de activación
- Nueva vista de monitorización de activación a nivel de zona en la aplicación web.
- Diseño y ejecución de pruebas de unitarias y de integración.

Se han diseñado dos algoritmos, ambos muy sencillos, para gestionar la activación del relé del actuador. Un primer algoritmo basado en *if ... else ...* con rangos de los valores de los sensores, y otro algoritmo un poco más preciso basado en lógica difusa.

Relativo al algoritmo de activación basado en lógica difusa, en este caso solo queremos saber si activar o desactivar (valor *crisp* o nítido). Para ello se

ha definido la salida del modelo como regar poco o regar mucho (en lógica normal, regar o no regar), considerando que en un futuro se pueda tener control sobre la electroválvula y mejorar la precisión de la salida del modelo.

Sprint 11: Integración III: Actuador - Servicio de procesamiento

Los objetivos de este *sprint* se detallan a continuación:

- Diseño y ejecución del plan de pruebas (se utilizan valores aleatorios generados por un simulador):
 - Comprobar que el servicio de procesamiento envía el evento de activación y recibe la confirmación de activación por parte del actuador.
 - Comprobar que el servicio de procesamiento envía el evento de desactivación y recibe la confirmación de desactivación por parte del actuador.
 - Comprobar que se almacenan los periodos de activación en la colección de la base de datos no relacional.
 - Comprobar que los datos de activación almacenados en la base de datos no relacional por parte del servicio de procesamiento se visualizan en la vista de monitorización (vista de las zonas.)
- Comprobar despliegue conjunto de aplicación web y servidor.

Sprint 12: Servicio de autenticación

Los objetivos de este *sprint* se detallan a continuación:

- Diseñar y desarrollar un sistema de autenticación basado en usuario y contraseña
 - Gestión de usuarios.
 - Gestión de contraseñas.
 - Gestión de *login* y *logout*.
 - Gestión de *access tokens*.
- Creación de un contenedor *Docker* para el despliegue.

Debido a las limitaciones de tiempo, en este servicio del sistema no se programaron pruebas unitarias ni de integración a nivel software, a diferencia del servidor central de riego. Las pruebas de funcionamiento se hicieron a mano utilizando *Postman* como herramienta de apoyo.

El servicio desarrollado se basa en la obtención de un *token* para ser utilizado por parte de la aplicación web para saber si un usuario está registrado o no y si se trata de un usuario administrador. La idea es extender el uso de los *tokens* de autenticación a las propias Apis REST del servicio central de riego.

Sprint 13: Pantalla de login y refactorización

Los objetivos de este *sprint* se detallan a continuación:

- Diseñar e implementar una vista de *login* en la aplicación web.
- Refactorizar el resto de vistas para limitar la funcionalidad permitida teniendo en cuenta de si el usuario es o no es un administrador.
 - No permitir crear, modificar o eliminar zonas y controladores a usuarios no administradores
 - El usuario no administrador simplemente puede visualizar zonas y controladores.

Se ha analizado la necesidad de estar *logueado* con un usuario válido para permitir a los usuarios finales entrar en la aplicación. La conclusión a la que se ha llegado, por motivos de limitar el acceso y de seguridad, es que el usuario final necesita un usuario. Por lo tanto, el usuario administrador del sistema se hará cargo de generar las credenciales de acceso.

Teniendo en cuenta el alcance del proyecto y el tiempo límite de desarrollo con el que se cuenta, se ha analizado la necesidad de disponer o no de una vista de gestión de usuarios en la aplicación web para darlos de alta. Se ha llegado a la conclusión de que esta funcionalidad no está alineada con los objetivos del proyecto y no aporta nada nuevo al prototipo, simplemente se trata de una funcionalidad que haría al sistema mejorar. Se crea el *ticket* correspondiente en *GitHub* para llevar a cabo en un futuro.

Sprint 14: Integración IV: Servicio de autenticación - Aplicación Web

Los objetivos de este *sprint* se detallan a continuación:

- Diseño y ejecución del plan de pruebas (se parte de datos almacenamos previamente en las bases de datos):
 - Comprobar que la aplicación te redirige a la vista de *login* al intentar acceder a la vista de monitorización de un controlador sin estar previamente logueado.
 - Comprobar la validación de credenciales en la vista de *login* y la correcta redirección.
 - Comparar las opciones de gestión sobre las zonas y los controladores entre un usuario que es administrador y otro que no lo es.
- Comprobar despliegue conjunto de aplicación web, servicio central de riego y servicio de autenticación.

Durante la realización de este *sprint* se detecta una funcionalidad no esencial que sería muy interesante llevar a cabo, una vista de monitorización resumen de cada zona. Se identifica esta nueva posible funcionalidad y se crea el *ticket* correspondiente en *GitHub* para llevar a cabo en un futuro.

Sprint 15: Documentación final

Los objetivos de este *sprint* se detallan a continuación:

- Documentar las Apis REST desarrolladas en los servicios de riego y de autenticación.
- Revisión de la documentación.
- Versión definitiva de los apartados principales con las correcciones indicadas.
- Limpieza del código fuente.

Durante los *sprints* previos se fueron cubriendo los diferentes apartados y avanzando en la redacción de la memoria para no acumular al final del proyecto todas las tareas de documentación.

Sprint 16: Anexos

Los objetivos de este *sprint* se detallan a continuación:

- Revisión de los anexos.
- Versión definitiva de los anexos con las correcciones indicadas.

Durante los *sprints* previos se fueron cubriendo los diferentes apartados y avanzando en la redacción de Anexos como el plan de proyecto, la especificación de requisitos o el diseño del sistema.

Sprint 17: Presentación

Los objetivos de este *sprint* se detallan a continuación:

- Creación de la presentación del TFM.
- Preparación y creación de la demostración del sistema.
- Grabación de los dos vídeos.
- *Commit* definitivo en el repositorio con la memoria y la presentación.

La realización de la demostración supuso el momento más complicado de todo el proyecto debido a la limitación de tiempo marcado en las bases del TFM. Se ha intentado mostrar los módulos más importantes del sistema y cómo funcionaría en un entorno real.

Planificación temporal

En este último subapartado se mostrará un diagrama temporal en donde se puede visualizar la planificación temporal estimada para cada uno de los *sprints*, con fecha de inicio y fin, y la ejecución real de cada uno de los *sprints* con el desfase. Se considera interesante este diagrama para apreciar la gestión de tiempo y posibles inconvenientes sucedidos a lo largo del proyecto.

En la Figura A.1, se puede apreciar esta planificación, destacando en color *verde* la estimación y en color *azul* los intervalos temporales necesarios para cumplir los objetivos de cada uno de los *Sprints*. EL diagrama se divide en semanas (se marca como día de inicio el lunes), y para cada semana se puede apreciar 7 divisiones que equivalen a los días de la semana.

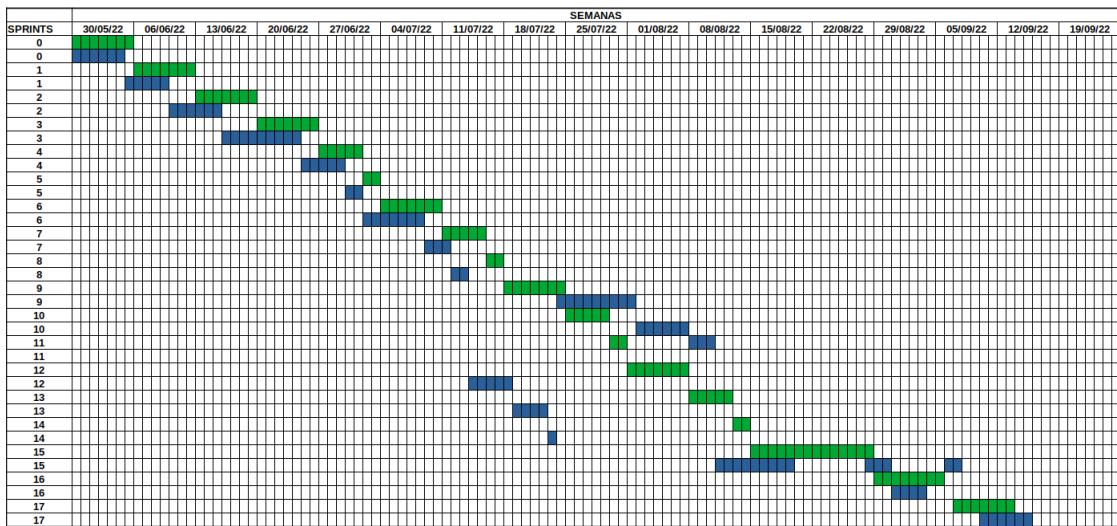


Figura A.1: Planificación temporal del proyecto

Se presentan a continuación una serie de conclusiones obtenidas al analizar el diagrama:

- Debido a las circunstancias logísticas, se decidió llevar a cabo el *Sprint 12* antes que el *Sprint 9*; esta decisión no ha supuesto ningún tipo de atraso en la planificación, al contrario, permitió adelantar trabajo posterior mientras se esperaba el microcontrolador encargado de la gestión de riego.
- Los *Sprints* relativos al inicio del proyecto, estudio del arte y el diseño del sistema, llevaron un algunos días menos de los previstos debido a la experiencia previa con proyectos de este tipo.
- Destacar el tiempo invertido en el *Sprint* relativo a los datos de captación de datos de sensorización debido a comunicación serie implementada entre el *Arduino UNO* y el *NodeMCU*.
- En cuanto al *Sprint* encargado de la autenticación, llevo algunos días a mayores de los planificados, debido a la necesidad de disponer de un servicio base lo suficientemente consolidado para futuras funcionalidades. Este tiempo se complementó con el tiempo ganado a la hora de implementar el actuador de riego (tiempo ganado gracias al uso de la *Raspberry Pi pico W*).
- Las tareas de documentación se fueron realizando a lo largo del todo el proyecto, acorde a los diferentes *Sprints* realizados (como diagramas,

especificación de requisitos); el *Sprint* de documentación se corresponde a la redacción final de la memoria y a su posterior formateo. Debido a las correcciones por parte de los tutores, el *sprint* se lleva a cabo a lo largo de varias semanas.

A.3. Viabilidad económica

Esta sección describe con cierto detalle los aspectos relativos a la viabilidad económica teniendo en cuenta los objetivos futuros de desplegar el sistema en un entorno físico de pruebas.

El análisis llevado a cabo en este apartado es considerando que la aplicación se va a lanzar en un entorno de pruebas previo a su posterior puesta en producción en diversas huertas. Para este último paso comentado, la puesta en producción en un entorno completamente real, será necesario llevar a cabo un análisis económico más detallado.

Requisitos tecnológicos

Se muestran a continuación el listado de requisitos tecnológicos necesarios para desarrollar este proyecto, asumiendo los conocimientos de herramientas, tecnologías y lenguajes de programación por parte del desarrollador:

- **Ordenador de desarrollo** que cumpla con los requisitos mínimos para ejecutar contenedores *Docker*: mínimo de 4GB de RAM y un procesador de gama media.
- **Raspberry Pi 4** que servirá como servidor central del prototipo.
- **Microcontroladores** para conectarse y leer los datos de los sensores y gestionar el relé.

Según el número de usuarios conectados y la cantidad de zonas y controladores gestionados, será necesario migrar el despliegue de los servicios centrales a un ordenador más potente para ser capaz de gestionar el incremento en el número de usuarios.

En cuanto a los conocimientos técnicos del desarrollador para desarrollar este sistema, se necesitarían cumplir los siguientes mínimos:

- Un año de experiencia en desarrollo de **aplicaciones web**. Conocimientos de *Vue*, *HTML*, *CSS* y *JavaScript*.

- Dos años de experiencia en el diseño y desarrollo de **sistemas asíncronos**. Conocimientos de sistemas basados en tareas y diseño y desarrollo de APIs REST en *Python*.
- Menos de un año de experiencia en programación de **microcontroladores** como *Arduino*.

Presupuesto del prototipo

En este apartado se realizará el proceso de análisis de costes cuyo objetivo principal es organizar el presupuesto del proyecto. Los costes que podemos encontrar en un proyecto son dos:

- Los costes que se pueden asignar de forma clara y definida al proyecto se denominan **directos**. Un ejemplo puede ser las horas trabajadas por cada trabajador.
- Los costes **indirectos** son los repartidos entre diferentes proyectos como puede ser el consumo de agua.

En los costes directos, debido a las características de este proyecto, incluiremos solo el sueldo del desarrollador como parte del equipo de desarrollo. Se considera el coste por hora igual a **15€/hora**.

En cuanto a los costes indirectos, es necesario incluir el servicio de Internet, la electricidad o el agua consumida por el alumno, entre otros. Como suele ser común en los proyectos software, y como se indica en el documento de costes indirectos de la USC¹, se tomará como el 21 % del coste final.

Un aspecto muy importante a tener en cuenta a la hora de analizar costes son los costes del equipo de trabajo de los desarrolladores. En este caso, al tratarse de un solo desarrollador, se tomará el coste del ordenador utilizado (1600 €) estimando una vida útil de 3 años (533 €) y un uso exclusivo de trabajo. Por lo tanto el coste del ordenador sería de **133 €** aproximadamente:

$$1600 \quad / \quad ((3 \times 12 \text{ meses}) \quad / \quad 3 \text{ meses de uso}) \quad = 133.33$$

En la tabla A.1 se muestra un resumen de los costes del proyecto.

¹http://imaisd.usc.es/ftp/oit/documentos/561_gl.pdf

Concepto	Precio Ud.	Unidades	Importe
<i>Raspberry Pi (pack)</i>	85	1	85
<i>Arduino Uno (pack)</i>	10	1	10
<i>Raspberry Pi pico w</i>	7	1	7
<i>Sensor FC-28</i>	1.75	2	2.5
<i>Sensor FC-37</i>	2.20	2	4.4
<i>Sensor LM-35</i>	1.75	2	2.5
<i>ESP8266</i>	2.75	2	5.5
<i>Relé 5V</i>	0.75	2	1.5
<i>Cableado</i>	10	1	10
<i>Portátil desarrollo</i>	133	1	133
<i>Analista Programador Python</i>	2250	1	2250
Subtotal			2511.4 €
<i>Costes indirectos (21 %)</i>			527.4
Precio total			3038.8 €

Tabla A.1: Tabla de costes

A.4. Viabilidad legal

Esta sección muestra los aspectos relativos a la viabilidad legal en el uso y aplicación del sistema desarrollado. El código fuente desarrollado cuenta con licencia MIT, lo que permite a cualquier desarrollador interesado acceder al código fuente y utilizarlo dentro de otro software propietario.

Entre los permisos con los que cuenta es importante destacar la posibilidad de su uso comercial, se puede modificar el código fuente y se puede distribuir sin ningún tipo de restricción ni penalización. En cuanto las responsabilidades, el autor del código fuente no se hace responsable de cualquier tipo de daño, reclamo u otro tipo de responsabilidad que puedan surgir por el uso del software.

Para finalizar, en cuanto a las condiciones de uso, es necesario incluir en todas las partes en dónde el software se está utilizando la marca personal de derechos exclusivos del autor.

Apéndice B

Especificación de Requisitos

B.1. Introducción

En este apartado se muestran los objetivos generales del TFM y el listado y especificación de los requisitos funcionales y no funcionales.

B.2. Objetivos generales

Tal y como se plantea en el Capítulo 2, el objetivo principal de este Trabajo Fin de Máster es el diseño e implementación de un **prototipo de sistema de gestión de riego** que permita la recogida, almacenamiento y monitorización de una serie de controladores que permiten la gestión de una serie de sensores que permitan la medición de valores medioambientales como puede ser la temperatura o la humedad, permitiendo la automatización y gestión de un sistema de riego asociado para gestionar el consumo de agua en una huerta de fresas.

Aunque actualmente el sistema permita la gestión de un sistema de riego, la idea es desarrollar el sistema pensando en poder gestionar diferentes sistemas y servicios mediante actuadores genéricos. La idea a medio y corto plazo es evolucionarlo a un **sistema de gestión automático de cultivos**.

B.3. Especificación de requisitos

En base a los objetivos fijados y a las reuniones realizadas, principalmente, en la primera fase del proyecto con los tutores del proyecto, se han identificado

una serie de requisitos que tendrá que cumplir la arquitectura desarrollada para que cumpla con los criterios de aceptación establecidos. Los requisitos identificados se dividen en dos grandes grupos: funcionales y no funcionales.

En los apartados posteriores se muestra unas tablas con las descripciones según el estándar IEEE [38] de cada uno de los requisitos identificados. Para su mejor clasificación, los requisitos funcionales se identifican con el identificador alfanumérico **RF-XX** (donde XX es el número secuencial que le corresponde) y los no funcionales con **RNF-XX**. En cuanto a la importancia de los requisitos se define la siguiente escala:

ESCALA DE IMPORTANCIA	
Vital	Imprescindible para considerar el proyecto completado.
Deseable	Aumenta la calidad del producto pero no es indispensable.

Requisitos funcionales

En este apartado se muestra el listado de requisitos funcionales identificados.

ID	RF-01
Nombre	Almacenar datos de sensores
Descripción	El sistema deberá de permitir el almacenamiento persistente de los datos en una base de datos organizados por controladores.
Importancia	Vital
Validación	Se almacena la información relativa a cada sensor de forma ordenada en un formato definido (documentos <i>JSON</i>).

ID	RF-02
Nombre	Almacenar datos de controladores
Descripción	El sistema deberá de permitir registrar y gestionar los controladores del sistema.
Importancia	Vital
Validación	Se almacena la información a los controladores en una base de datos.

ID	RF-03
Nombre	Almacenar datos de zonas
Descripción	El sistema deberá de permitir registrar y gestionar las diferentes zonas que permiten agrupar los controladores
Importancia	Vital
Validación	Se almacena la información a las zonas en una base de datos.

ID	RF-04
Nombre	Almacenar datos de activación
Descripción	El sistema deberá de permitir el almacenamiento persistente de los periodos de activación del actuador en un formato definido (documentos <i>JSON</i>).
Importancia	Deseable
Validación	Se almacena la información relativa a los periodos de activación del actuador.

ID	RF-05
Nombre	Recoger datos de sensores
Descripción	El sistema deberá de permitir la recogida de los datos de los sensores de temperatura, humedad y lluvia utilizando un sistema empotrado como controlador.
Importancia	Vital
Validación	Lectura correcta de cada uno de los sensores.

ID	RF-06
Nombre	Enviar datos de sensores
Descripción	El sistema deberá de permitir el envío de los datos desde el microcontrolador al servidor central utilizando algún tipo de comunicación asíncrona y un formato definido (formato <i>JSON</i>).
Importancia	Vital
Validación	Envío de los valores de sensores siguiendo un formato específico por medio de un sistema de comunicación asíncrona.

ID	RF-07
Nombre	Gestionar el sistema el sistema de riego
Descripción	El sistema deberá de permitir definir un módulo genérico que permita la activación de diferentes sistemas electrónico-mecánicos mediante comunicación asíncrona.
Importancia	Deseable
Validación	Activación y desactivación del actuador.

ID	RF-08
Nombre	Algoritmo de activación
Descripción	El sistema deberá de definir un algoritmo de activación a partir de los datos recibidos de los sensores de los controladores.
Importancia	Deseable
Validación	Dados unos valores de entrada se obtiene como resultado si se activa o no.

ID	RF-09
Nombre	Visualización en tiempo real de los datos de sensores
Descripción	El sistema deberá de estar provisto de una interfaz para la visualización de los datos en tiempo real de los sensores.
Importancia	Vital
Validación	El sistema mostrará al usuario los datos leídos por los sensores en formato de gráfica.

ID	RF-10
Nombre	Visualización del histórico de los datos de sensores
Descripción	El sistema deberá de estar provisto de una interfaz para la visualización de los datos históricos de los sensores para un rango temporal.
Importancia	Deseable
Validación	El sistema mostrará al usuario los datos leídos por los sensores en formato de gráfica.

ID	RF-11
Nombre	Visualización del histórico de los rangos de activación
Descripción	El sistema deberá de estar provisto de una interfaz para la visualización de los datos históricos de los rangos de activación.
Importancia	Deseable
Validación	El sistema mostrará al usuario los rangos de activación de los actuadores en formato de gráfica.

Requisitos no funcionales

En este apartado se muestra el listado de requisitos no funcionales identificados.

ID	RNF-01
Tipo	Fiabilidad
Nombre	Aplicación multiusuario
Descripción	El sistema deberá permitir dar servicio a varios usuarios de forma simultánea en la aplicación.
Importancia	Vital

ID	RNF-02
Tipo	Entrega
Nombre	Documentación técnica simple
Descripción	La documentación técnica (manuales de usuario y manual de instalación) debe de ser simple y lo más completa posible para que futuros desarrolladores no inviertan tiempo innecesario en el proceso.
Importancia	Vital

ID	RNF-03
Tipo	Usabilidad
Nombre	Aplicación web usable
Descripción	El sistema deberá presentar al usuario una estructura que permita usarla de forma sencilla sin provocar pérdidas de tiempo.
Importancia	Vital

ID	RNF-04
Tipo	Entrega
Nombre	Tiempo máxima de desarrollo de 4 meses
Descripción	El proyecto debe de estar terminado en un plazo de 4 meses (finalización a mediados de septiembre).
Importancia	Vital

ID	RNF-05
Tipo	Estándares
Nombre	Herramientas OpenSource
Descripción	Las herramientas utilizadas durante el desarrollo deben de ser gratuitas (o en su defecto utilizar versiones gratuitas) y de código abierto.
Importancia	Deseable

ID	RNF-06
Tipo	Estándares
Nombre	Uso de protocolos estándar
Descripción	La comunicación entre los diferentes módulos tiene que ser utilizando protocolos HTTP o HTTPS.
Importancia	Deseable

ID	RNF-07
Tipo	Herramientas
Nombre	Framework JavaScript
Descripción	Utilizar un framework JavaScript actual para el desarrollo de la aplicación web.
Importancia	Vital

ID	RNF-08
Tipo	Comunicación
Nombre	Comunicación transparente
Descripción	El proceso de comunicación entre los diferentes módulos que forma el sistema tiene que ser transparente al usuario final.
Importancia	Vital

ID	RNF-09
Tipo	Usabilidad
Nombre	Restringir funcionalidades en la aplicación
Descripción	Los usuarios que se identifiquen en la aplicación sin ser administradores no tendrán acceso a las funcionalidades de gestión u otras funcionalidades sensibles.
Importancia	Vital

ID	RNF-10
Tipo	Rendimiento
Nombre	Escalabilidad vertical y horizontal (almacenamiento)
Descripción	El sistema de elegir un gestor de almacenamiento eficiente teniendo en cuenta el tipo de dato a almacenar.
Importancia	Vital

Apéndice C

Especificación de diseño

Este apartado muestra las diferentes partes del diseño llevado a cabo para el sistema. Para el modelado del software se hará uso de *UML* [39], concretamente de los diagramas de clases, interacción y despliegue.

Se mostrarán la estructura de las bases de datos, diagramas para la compresión de la estructura y funcionamiento del sistema así como bocetos de la interfaz web deseada.

C.1. Diagrama de Casos de Uso

Una vez identificados los diferentes requisitos del proyecto, es necesario describir los diferentes casos de uso del sistema. Los casos de uso sirven para describir los posibles escenarios con los que se encontrarán los diferentes actores de la aplicación. En este caso, para simplificar la especificación de los casos de uso se identifica como único actor del sistema el usuario final, con y sin privilegios, de la aplicación web.

El formato para identificar los casos de uso seguirán un formato parecido a los requisitos, **CU-XX**. En la Figura C.1 se muestra el diagrama con los casos de uso identificador llevados a cabo por el usuario final.

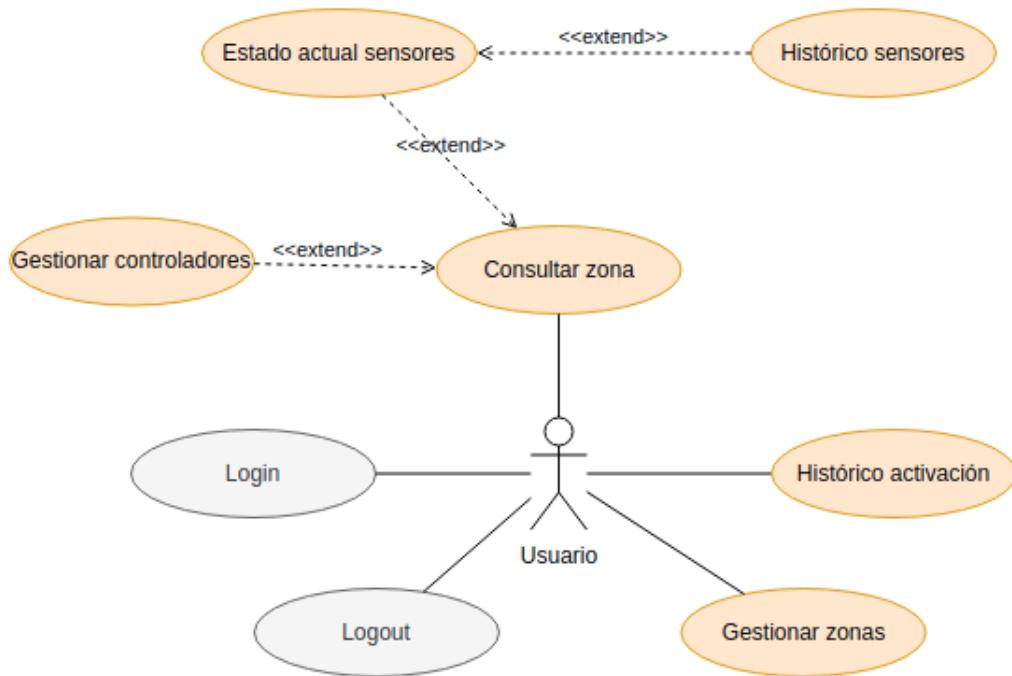


Figura C.1: Diagrama de Casos de Uso

ID	CU-01	
Nombre	Login	
Descripción	Permite a un usuario validar unas credenciales en el sistema para poder entrar a la página de inicio de la aplicación.	
Precondición	Dirigirse a la ruta del servidor web donde está alojada el sistema.	
Secuencia normal	Paso	Acción
	1	Escribir el nombre de usuario.
	2	Escribir la contraseña del usuario.
Postcondición	Se redirige al usuario a la página de inicio del sistema.	
	Paso	Acción
Excepciones	3	Credenciales ingresadas incorrectas.
Importancia	Deseable	

Tabla C.1: Caso de Uso Login

ID	CU-02	
Nombre	Logout	
Descripción	Permite a un usuario salir de la sesión actual de la aplicación web y volver a la página de login.	
Precondición	CU-01	
Secuencia normal	Paso	Acción
	1	Pulsar botón de Logout (o ícono correspondiente).
Postcondición	Se redirige al usuario a la página de login para ingresar de nuevo las credenciales.	
Importancia	Deseable	

Tabla C.2: Caso de Uso Logout

ID	CU-03	
Nombre	Consultar zona	
Descripción	Permite a un usuario obtener información adicional de una zona de la lista de zonas de la página de inicio.	
Precondición	CU-01	
Secuencia normal	Paso	Acción
	1	Se obtiene información relativa a los controladores.
	2	El usuario puede ejecutar CU-04 (Información del estado de los sensores).
Postcondición	Se muestran nuevas opciones en la aplicación web al usuario final (mediante un desplegable, tabla expansiva, modal o una nueva vista).	
Importancia	Vital	

Tabla C.3: Caso de Uso Consultar Zona

ID	CU-04	
Nombre	Estado actual sensores	
Descripción	Permite a un usuario obtener información en tiempo real de los valores recibidos de los sensores por parte del controlador seleccionado.	
Precondición	CU-01 y CU-03	
Secuencia normal	Paso	Acción
	1	Se pulsa sobre el botón de <i>Visualizar el estado</i> o el ícono equivalente
Postcondición	Se muestran la información en tiempo real de los valores de sensores del controlador seleccionado en formato gráfico.	
Importancia	Vital	

Tabla C.4: Caso de Uso Estado Actual Sensores

ID	CU-05	
Nombre	Histórico sensores	
Descripción	Permite a un usuario visualizar los datos de los sensores de un controlador en un intervalo temporal determinado.	
Precondición	CU-01 y CU-04	
Secuencia normal	Paso	Acción
	1	Se especifica una fecha de inicio.
	2	Se especifica una fecha de fin.
	3	Se pulsa el botón <i>Buscar</i>
Postcondición	Se muestran la información de los sensores en formato gráfico. El resultado es muy parecido al obtenido en CU-04, lo que cambian son los datos a mostrar.	
Importancia	Deseable	

Tabla C.5: Caso de Uso Histórico Sensores

ID	CU-06	
Nombre	Histórico activación	
Descripción	Permite a un usuario visualizar los intervalos de activación del actuador asociado a la zona	
Precondición	CU-01	
Secuencia normal	Paso	Acción
	1	El usuario visualiza el listado de todas las zonas que forman parte de la aplicación.
	2	El usuario pulsa sobre el botón de <i>Visualizar histórico de activación</i> o en ícono equivalente.
	3	Se muestra un formulario formado por fecha inicio y fecha fin.
	4	Se especifica una fecha de inicio.
	5	Se especifica una fecha de fin.
	3	Se pulsa el botón <i>Buscar</i>
Postcondición	Se muestran el conjunto de intervalos de activación (fecha inicio y fecha fin) en formato gráfico.	
Importancia	Deseable	

Tabla C.6: Caso de Uso Histórico Activación

ID	CU-07	
Nombre	Gestionar zonas	
Descripción	El sistema debe permitir al usuario permitir crear, eliminar, modificar y visualizar la información de zonas (<i>CRUD</i>).	
Precondición	CU-01	
Secuencia normal	Paso	Acción
	1	El usuario visualiza el listado de todas las zonas que forman parte de la aplicación.
	2.a	El usuario pulsa el botón para eliminar una zona determinada y el sistema le pide confirmación mediante un mensaje de alerta.
	2.b	El usuario pulsa el botón para editar una zona determinada y el sistema muestra la información actual de la zona y un formulario que permite modificar ciertas propiedades.
	2.b (II)	El usuario modifica unas propiedades determinadas y pulsa el botón para guardar los cambios.
	2.c	El usuario pulsa el botón para registrar una nueva zona mostrando al usuario el mismo formulario que en el caso 2.b para cubrirlo.
	2.c (II)	El usuario cubre el formulario y pulsa el botón para registrar la nueva zona.
Postcondición	<p>El sistema redirige al usuario a la página de inicio volviendo a mostrar el listado de zonas registradas en el sistema.</p> <ul style="list-style-type: none"> ■ En el caso de eliminar una zona, en el listado no volverá a aparecer esa zona. ■ En el caso de registrar una zona, en el listado aparecerá esa zona. ■ En el caso de modificar una zona, el listado se mantendrá igual. 	
Excepciones	Paso	Acción
	2.c (II)	Ya existe una zona con el mismo nombre identificativo.
Importancia	Vital	

Tabla C.7: Caso de Uso Gestionar Zonas

ID	CU-08	
Nombre	Gestionar controladores	
Descripción	El sistema debe permitir al usuario permitir crear, eliminar, modificar y visualizar la información de los controladores (<i>CRUD</i>).	
Precondición	CU-01 y CU-03.	
Secuencia normal	Paso	Acción
	1	El usuario visualiza el listado de todos los controladores que están asignados a la zona.
	2.a	El usuario pulsa el botón para eliminar un controlador determinado y el sistema le pide confirmación mediante un mensaje de alerta.
	2.b	El usuario pulsa el botón para editar un controlador determinado y el sistema muestra la información actual del controlador y un formulario que permite modificar ciertas propiedades.
	2.b (II)	El usuario modifica unas propiedades determinadas y pulsa el botón para guardar los cambios.
	2.c	El usuario pulsa el botón para registrar un nuevo controlador mostrando al usuario el mismo formulario que en el caso 2.b para cubrirlo.
	2.c (II)	El usuario cubre el formulario y pulsa el botón para registrar el nuevo controlador.
Postcondición	<p>El sistema redirige al usuario a la página de inicio volviendo a mostrar el listado de controladores de la zona previamente seleccionada.</p> <ul style="list-style-type: none"> ■ En el caso de eliminar un controlador, en el listado no volverá a aparecer el controlador. ■ En el caso de registrar un controlador, en el listado aparecerá el nuevo controlador registrado. ■ En el caso de modificar un controlador, el listado se mantendrá igual. 	
Excepciones	Paso	Acción
	2.c (II)	Ya existe un controlador con el mismo nombre identificativo.
Importancia	Vital	

Tabla C.8: Caso de Uso Gestionar Controladores

C.2. Diseño de datos

Se muestra cómo se plantea la estructura de los datos en el sistema. Al contar con dos paradigmas de almacenamiento diferentes (relacional y no relacional), se divide este apartado en dos subapartados diferentes.

Base de Datos Relacional

En este subapartado se mostrarán los modelos de entidad-relación y el modelo relacionados de las dos bases de datos relacionales que forman parte del sistema: **gestión de controladores** y **gestión de usuarios**.

Diagrama Entidad-Relación

La información almacenada en esta base de datos es la relativa a los controladores que forman parte de la aplicación principalmente. Debido a que se consideraba necesario organizar los controladores por parcela de cultiva o *zonas* específicas, se decide crear una entidad de mayor peso encargada de ese rol y que puede tener diferentes tipos de funcionamiento asociados¹. En la Figura C.2 muestra la relación de las tres entidades descritas.



Figura C.2: Diagrama Entidad-Relación Controladores

Las zonas son las encargadas de gestionar 0 o muchos controladores (cardinalidad 0 a N), por lo que los controladores se puede considerar una entidad débil de las zonas. Cada una de las zonas tiene que tener un tipo de zona asociado (cardinalidad 1 a N).

Por último, es necesario gestionar los usuarios de la aplicación así como los tokens de acceso para cada usuario, todo ello estará almacenado en una base de datos distinta. En la Figura C.3 muestra la relación entre las dos entidades descritas.

La relación descrita es bastante sencilla, un usuario puede tener o no uno o varios tokens de acceso, por lo que la cardinalidad es la de 0 a N.

¹Funcionalidad de cara al futuro, actualmente solo se tienen en cuenta zonas de monitorización (sin actuadores) y zona de riego.

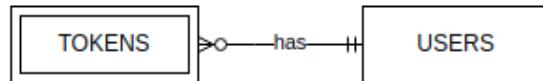


Figura C.3: Diagrama Entidad-Relación Usuarios

Modelo Relacional

Se muestra en las Figuras C.4 y C.5 los modelos relationales obtenidos de los modelos Entidad-Relación descritos en el apartado anterior.

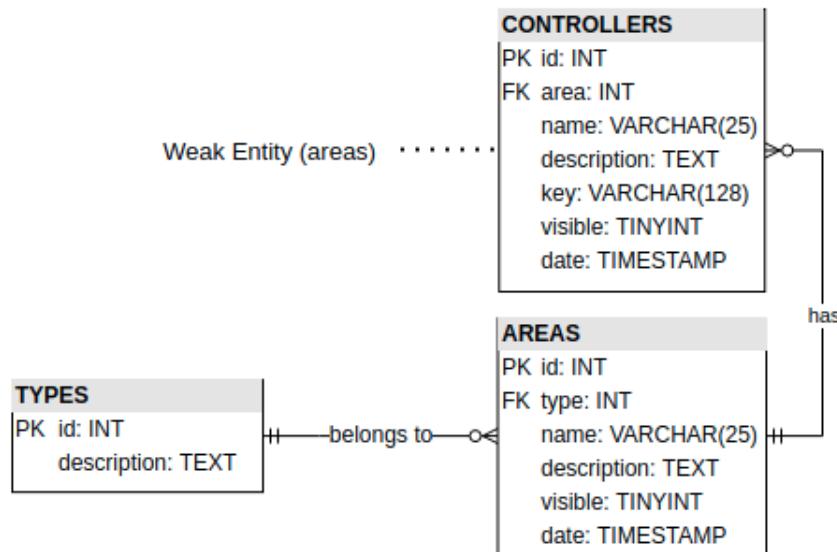


Figura C.4: Diagrama Relacional Controladores

En el caso de la **gestión de controladores**, la base de datos está formada por 3 tablas:

- **Tipos:** se almacena un listado de identificadores de tipos de zonas con su descripción. La acción llevada a cabo por cada tipo de zona es gestionada en el servicio central.
- **Áreas:** se almacena la información relativa a las zonas que organizan un conjunto de controladores. El atributo *name* tiene que ser único para todo el sistema. De forma obligatoria se tiene que asociar un tipo de zona que permitirá ejecutar un algoritmo u otro a la hora de gestionar los actuadores.

- **Controladores:** se almacena la información relativa a los diferentes controladores que forman cada una de las zonas registradas. El atributo *name* tiene que ser único para todo el sistema.

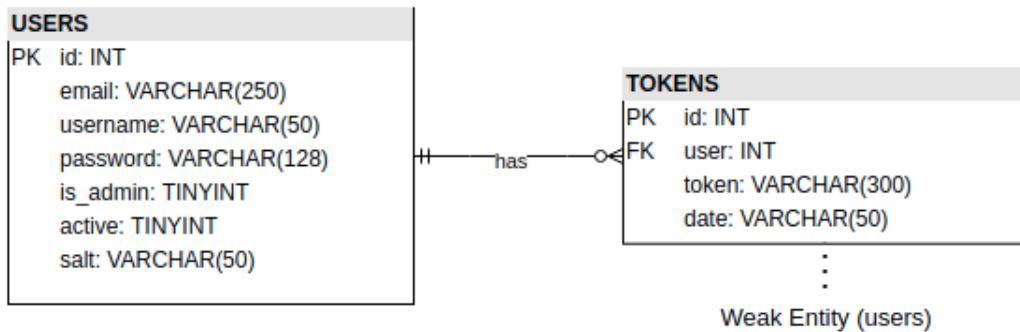


Figura C.5: Diagrama Relacional Usuarios

En el caso de la **gestión de usuarios**, la base de datos está formada por 2 tablas:

- **Usuarios:** se almacena información relativa a los usuarios registrados en el sistema como nombre de usuario y contraseña. También se almacena si cada usuario se trata de un administrador (acceso a funcionalidades restringidas) o si se encuentra el usuario activo.
- **Tokens:** se almacenan los diferentes tokens de acceso generados para cada uno de los usuarios registrados. Se guarda el propio token JWT gestionado por el servicio de autenticación del sistema.

Base de Datos no Relacional

En este subapartado se mostrarán las colecciones almacenadas en la base de datos no relacional: **histórico de datos de sensores** e **histórico de activación**. La Figura C.6 muestra el formato de los archivos *JSON*².

SENSORS_HISTORIC	IRRIGATION_HISTORIC
<pre>{ "controller_id": "<id_controlador>", "controller": "<nombre_controlador>", "area_id": "<id_zona>", "area": "<nombre_zona>", "humidity": "<valor_humedad>", "raining": "<true false>", "temperature": "<valor_temperatura>", "date": "<valor_fecha>", }</pre>	<pre>{ "area_id": "<id_zona>", "area": "<nombre_zona>", "irrigate": "<true false>", "start_date": "<valor_fecha>", "end_date": "<valor_fecha>", }</pre>

Figura C.6: Colecciones Histórico

La base de datos va a contar con 2 colecciones diferentes:

- **Sensores:** permite almacenar documentos con los datos de cada uno de los sensores gestionados por un controlador, en una zona determinada. De forma adicional, es obligatorio almacenar la fecha de inserción en la base de datos utilizada para, en la aplicación web, poder mostrar los datos.
- **Activación:** permite almacenar documentos con los intervalos de activación (fecha inicio y fecha fin) del actuador asociado a una zona determinada.

²Se ha escogido como gestor MongoDB, que permite almacenar en diferentes colecciones documentos en formato *JSON*.

C.3. Diagrama de Arquitectura

En este apartado se pretende mostrar el esquema de componentes adoptado para interconectas los diferentes elementos que forman parte del sistema de una manera muy visual y fácil de entender. La Figura C.7 muestra un esquema muy visual que no sigue el formato UML.

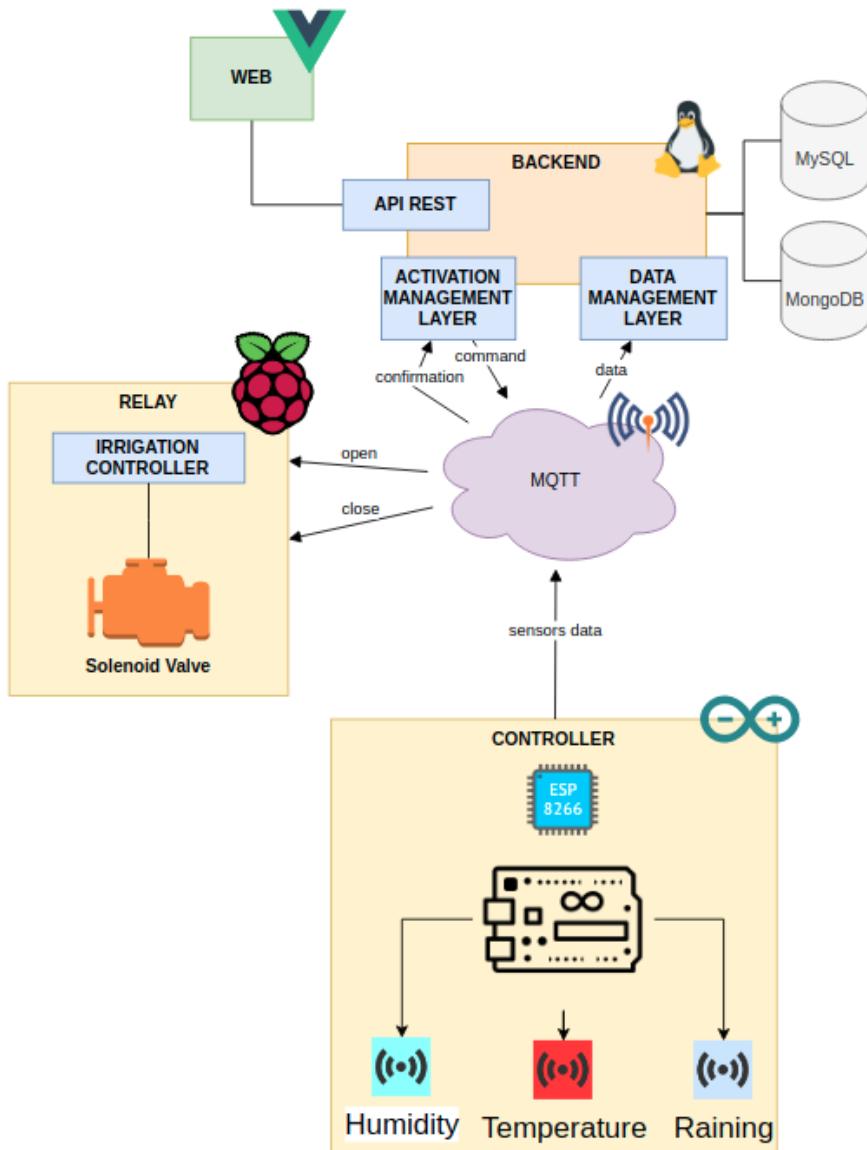


Figura C.7: Diagrama de arquitectura del sistema

La Figura C.8 muestra los diferentes módulos que forman el sistema. Como se puede observar, se distinguen 3 sistemas diferenciados:

- Uno sistema formado por los **microcontroladores**.
- Un sistema intermedio de la **recogida y procesado de los datos**.
- Un sistema superior, encargado de la comunicación con el usuario, en dónde se encuentra la **aplicación web**.

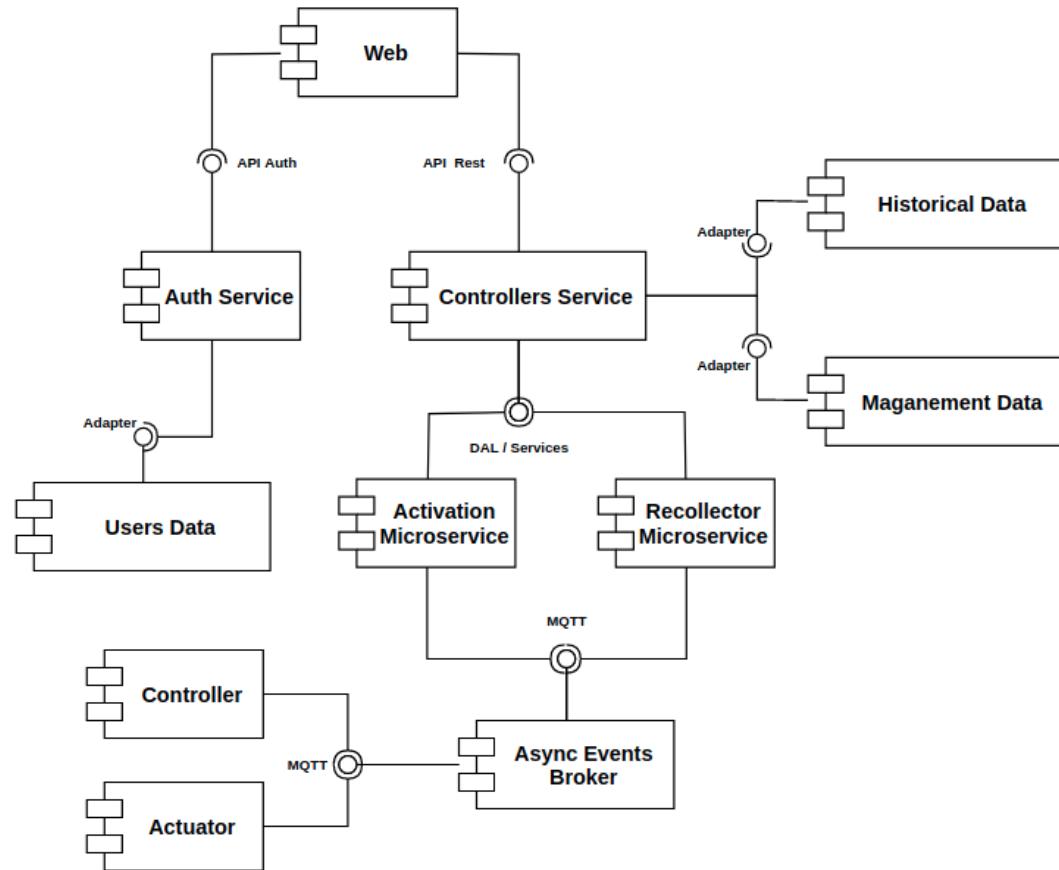


Figura C.8: Módulos del sistema

Se consideró el uso de este último diagrama debido a la complejidad de este problema, y a la necesidad de mostrar al usuario lo complejo que puede llegar a ser el sistema.

C.4. Diagramas de Secuencia

En este apartado se mostrarán los diagramas de secuencia que permitirán entender las comunicaciones y ejecución de funcionalidades entre los diferentes componentes que forman el sistema. Debido a la gran cantidad de diagramas que se podrían mostrar del sistema, se han escogido aquellos que involucren un mayor número de módulos así como una cierta complejidad. De todas formas, todos son muy sencillos de entender y seguir a simple vista.

En la Figura C.9 se puede apreciar la interacción por parte del usuario con la aplicación web para poder visualizar el estado de los sensores de un controlador y poder visualizar los datos en un intervalo concreto.

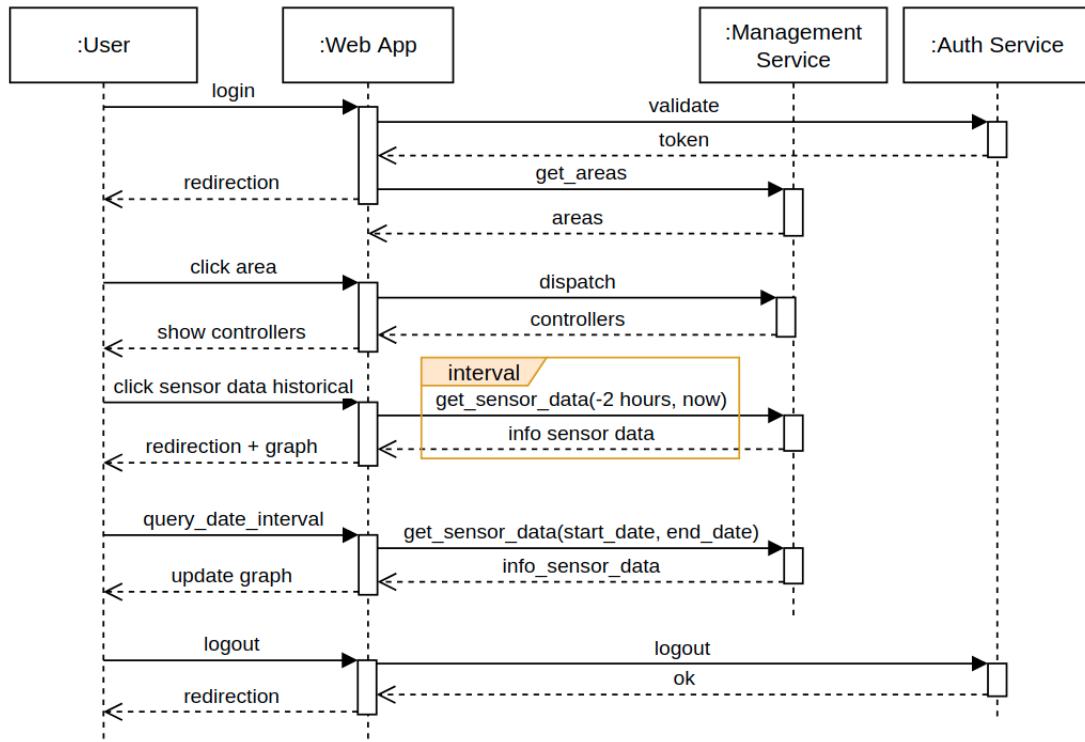


Figura C.9: Diagrama de Secuencia: Visualizar histórico de sensores

De este diagrama, simplemente destacar el cuadrado de *interval*, que indica que una vez el usuario entra en la vista de estado actual de los sensores, si el usuario dejara la aplicación web en el navegador, los datos se irían actualizando cada *X* segundos.

En caso de llevar a cabo una búsqueda por intervalos temporales, para visualizar información del histórico, esta actualización periódica se desactivará para simplemente mostrar los valores de búsqueda.

La Figura C.10 se puede apreciar un proceso completo de gestión de controladores por parte de un usuario, creando un controlador y luego eliminándolo, y finalizando editando un controlador del listado.

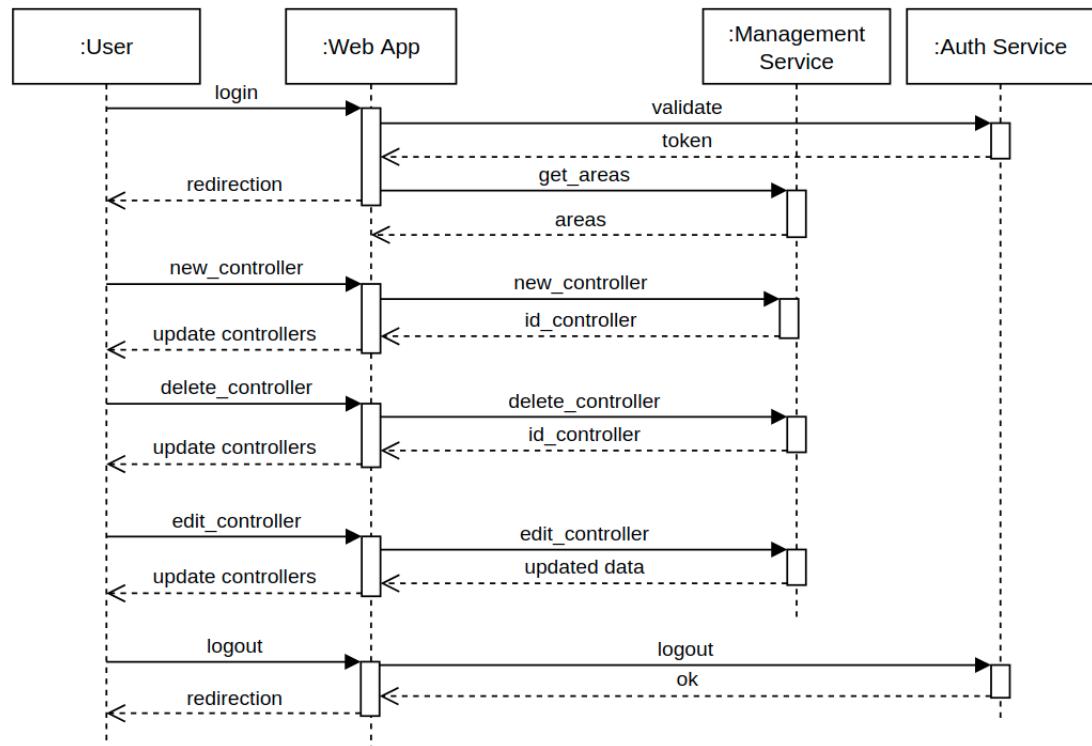


Figura C.10: Diagrama de Secuencia: Gestión de controladores

En el caso de la gestión de zonas, el diagrama resultado sería del mismo formato, cambiando simplemente el nombre de las funciones y las acciones.

El último diagrama que se va a mostrar en este apartado es el diagrama de secuencia de un microcontrolador para mostrar la interacción de los elementos hardware. La Figura C.11 muestra el diagrama del controlador debido a la gran cantidad de comunicaciones diferentes que tiene que llevar a cabo, además de estar formado por dos partes bien diferenciadas: lectura de datos y envío de datos.

Al igual que en el primer diagrama este diagrama cuenta con un par de cuadro de *interval* indicando la repetición periódica de las acciones de lectura de datos (*Arduino*) y del envío de datos por medio del *broker* (*NodeMCU*).

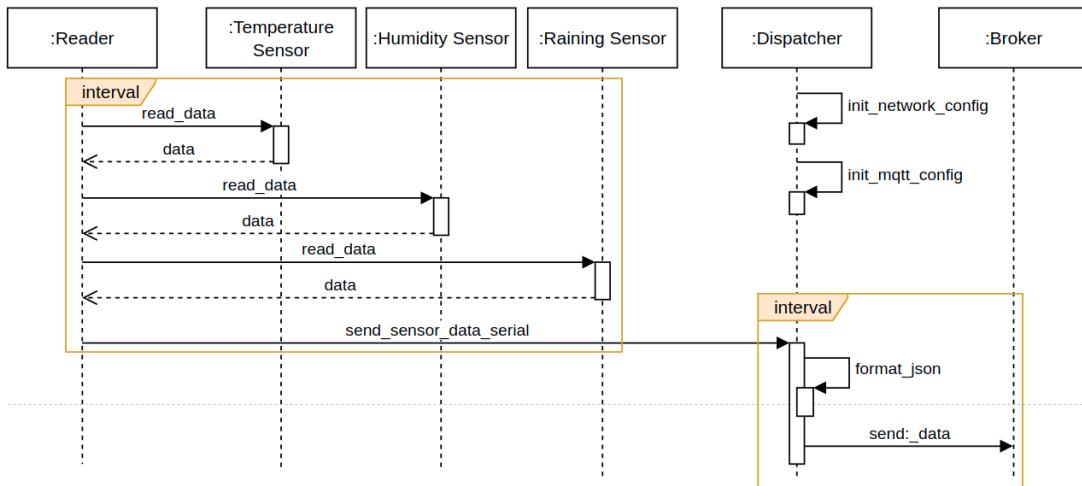


Figura C.11: Diagrama de Secuencia: Controlador Sensores

Para concluir este apartado, se comentará un par de aclaraciones y temas relativos a los diagramas de secuencia:

- No se muestran los diagramas de secuencia de los microservicios encargados de la recogida y almacenamiento de datos en la base de datos y de procesamiento y gestión de la activación de los actuadores porque la complejidad es muy sencilla. En un caso se recibe un mensaje del *broker* y llama a los servicios encargados del almacenamiento; en el otro llama a los mismos servicios para la lectura de datos y emite un evento al *broker*.
- En el caso del actuador, el diagrama era mucho más sencillo comparado con el controlador, por lo que no se ha considerado necesario. Este microcontrolador recibe un evento del *broker*, interactúa con el *relé* y emite un evento de confirmación de vuelta.

C.5. Diagrama de Despliegue

Se ha utilizado un despliegue basado principalmente en contenedores *Docker* en el servidor central. En cambio, los servicios desplegados en los diferentes microcontroladores (*actuador* y *controlador*), al no poder hacer uso de *Docker* debido a que el paradigma de despliegue y programación utilizado en estos sistemas es diferente, se han basado en la compilación del programa y la carga en memoria principal del microcontrolador. La Figura C.12 muestra el diagrama de despliegue diseñado, en donde se pueden apreciar los contenedores *Docker* en cajas de color amarillo y los microcontroladores en cajas de color gris.

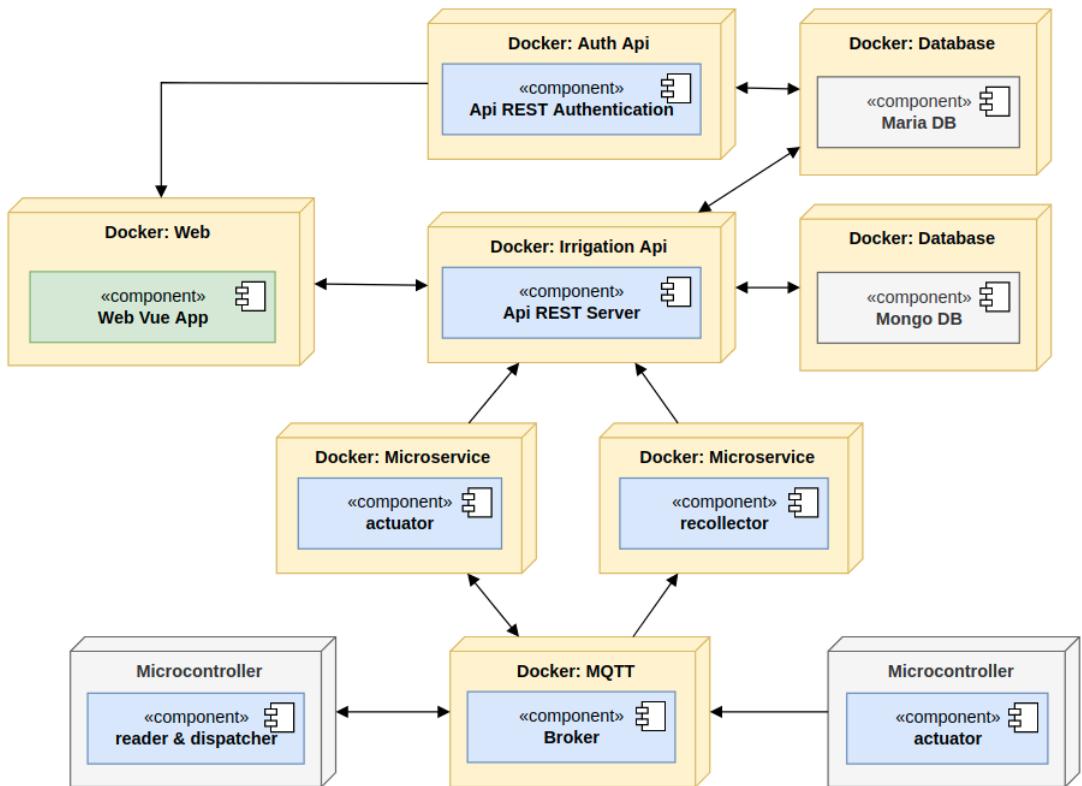


Figura C.12: Diagrama de Despliegue

C.6. Diseño de la Interfaz

Para diseñar los *mockups* y bocetos de la aplicación web se ha tenido en cuenta aspectos como la usabilidad, la sencillez y el minimalismo.

Login

La primera vista que se va a encontrar el usuario la primera vez que acceda a la URL de la aplicación web es un formulario para poder ingresar las credenciales de acceso.

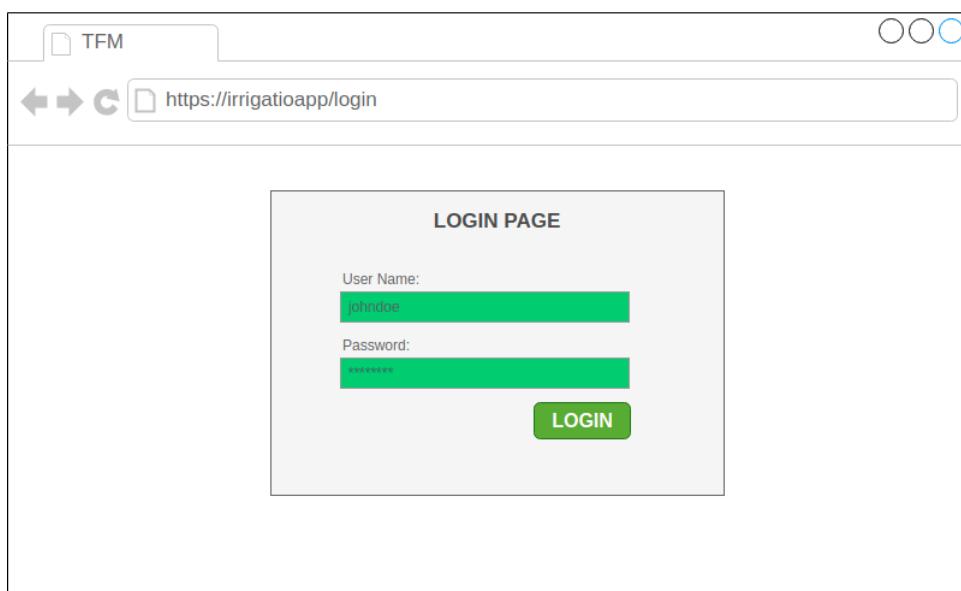


Figura C.13: Boceto: Página de Login

Home

Una vez el usuario ha validado sus credenciales contra el servicio de autenticación, se le mostrará la *home page* con el listado de zonas registradas en la aplicación, tal y como se puede apreciar en la Figura C.14. Adicionalmente, sobre cada una de las zonas registradas el usuario puede llevar ciertas acciones de gestión como puede ser modificarla o eliminarla. En esta misma página se mostrará la opción de registrar nuevas zonas al sistema.

Para finalizar la descripción de esta vista, comentar la opción de *consultar zona* que permite mostrar el listado de los controladores asociados en modo de tabla desplegada o algún componente web parecido.

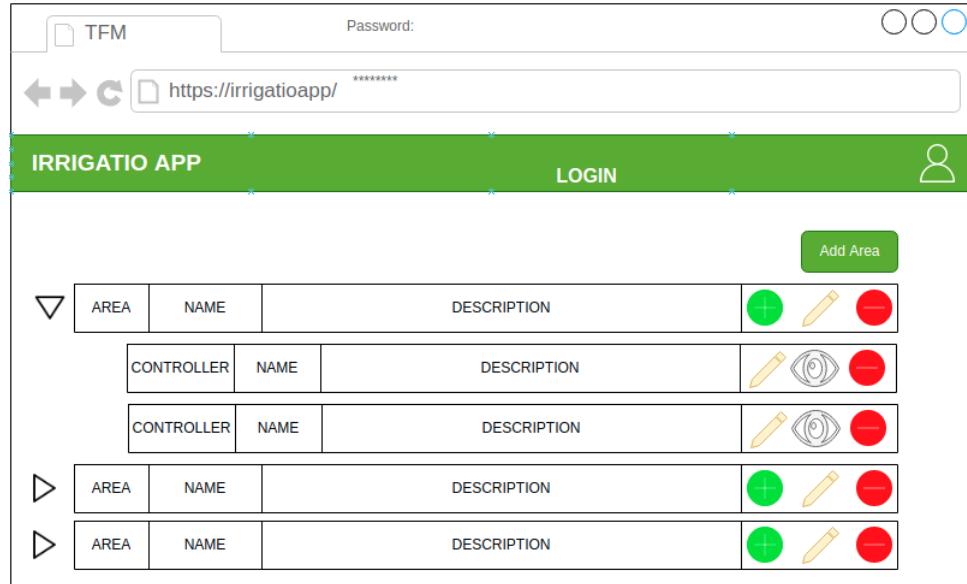


Figura C.14: Boceto: Página de Inicio

Gestión de Zonas

La Figura C.15 muestra los formularios en caso de querer crear una nueva zona o editar una zona existente.

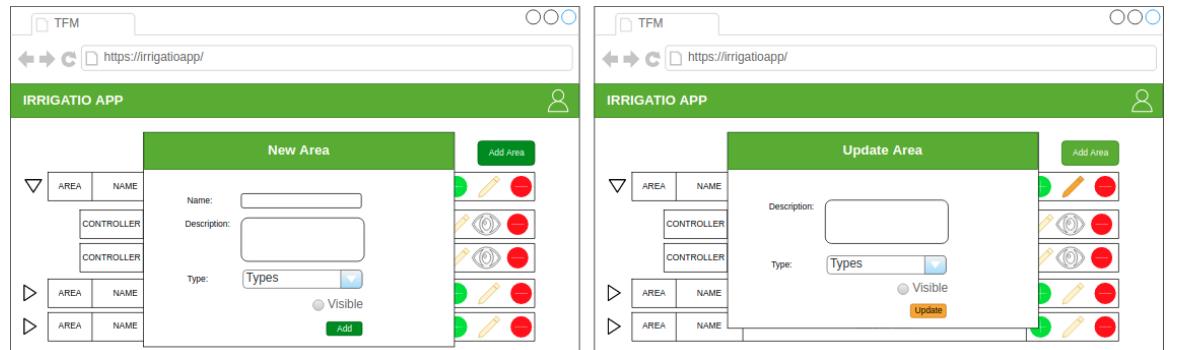


Figura C.15: Boceto: Gestión de Zonas

Gestión de Controladores

La Figura C.16 muestra los formularios en caso de querer crear un nuevo controlador o editar uno existente.

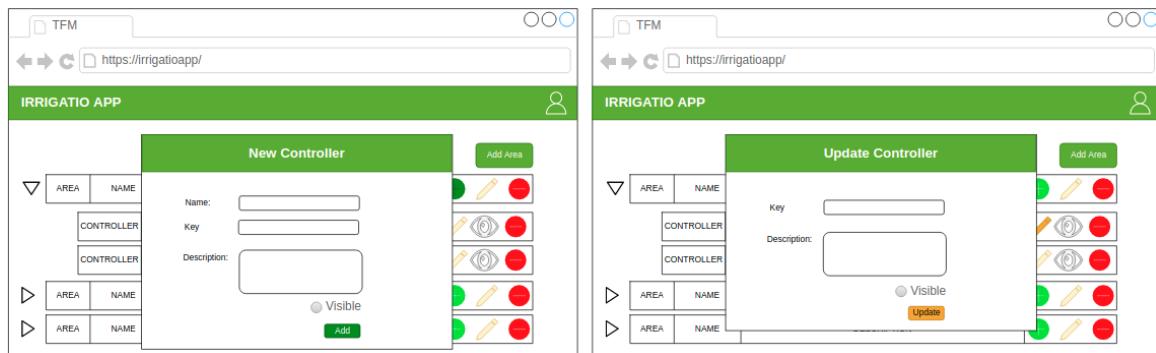


Figura C.16: Boceto: Gestión de Controladores

Monitorización

La última vista de la que se ha hecho un boceto es la de monitorización. Esta vista, que se puede apreciar en la Figura C.17, será igual tanto para monitorizar el estado actual e histórico de los sensores de un controlador determinado o los intervalos de activación del actuador.

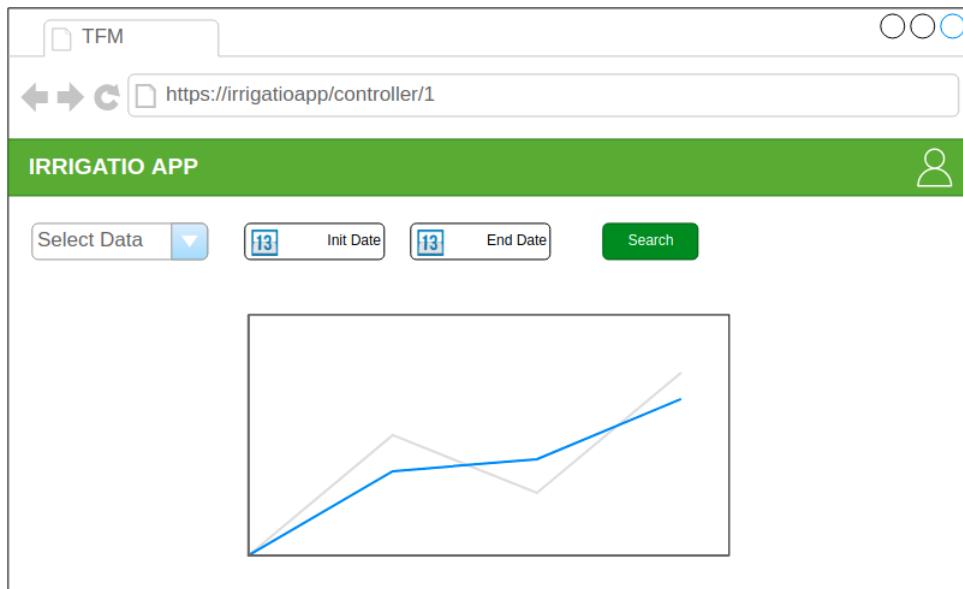


Figura C.17: Boceto: Visualización de histórico de datos

Se puede ver un formulario para indicar el intervalo temporal de búsqueda y una zona en donde se mostrarán los datos en formato gráfico.

C.7. Diseño circuito eléctrico

En este apartado se mostrará los planos de conexión de los diferentes elementos hardware que forman el proyecto. Como ya se ha introducido a lo largo de este documento, el sistema cuenta con dos microcontroladores distintos, uno centrado en la recogida y envío de datos al servidor, y otro microcontrolador encargado de gestionar la activación del relé.

El diagrama de conexiones entre el *Arduino UNO* y los sensores de humedad, temperatura y lluvia, y la conexión con el módulo *NodeMCU ESP8266*, se puede visualizar en la Figura C.18.

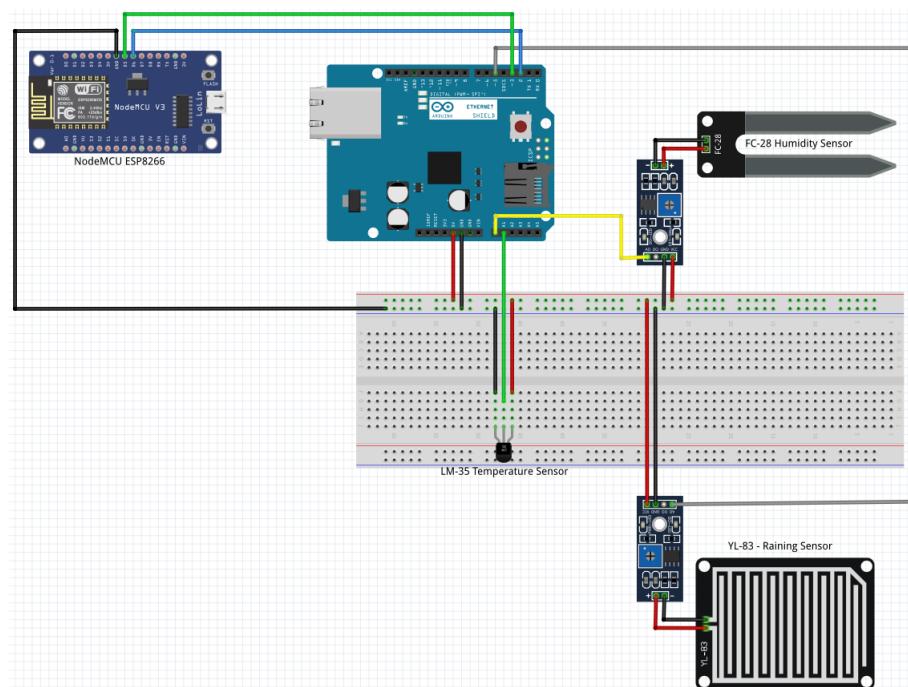


Figura C.18: Diagrama de conexiones del Controlador

Se puede apreciar que tanto el sensor de temperatura como el de humedad se conectan a los *pinouts* analógicos del *Arduino*, mientras que el sensor de lluvia a uno de tipo digital. En cuanto a la comunicación *Arduino UNO* y módulo *NodeMCU*, es de tipo serie.

Por último, el diagrama de conexiones entre la *Raspberry Pi pico W* y el *relé*, se puede visualizar en la Figura C.19.

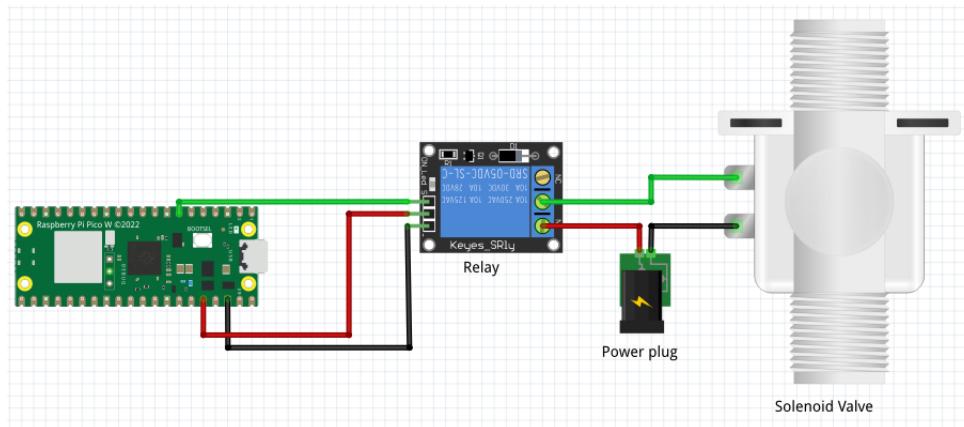


Figura C.19: Diagrama de conexiones del Actuador

De forma muy resumida, lo que se está viendo en la imagen simplemente es un *bypass* que permite cortar el flujo de corriente desde la fuente de potencia a la electroválvula mediante el uso de un *relé*.

Debido a las limitaciones del entorno de desarrollo, todas las pruebas llevadas a cabo y la demostración se han llevado a cabo utilizando un *led* como activador que simularía el comportamiento de la electroválvula. En la Figura C.20 se muestra el diagrama de conexiones, que es muy similar al mostrado en la Figura C.19.

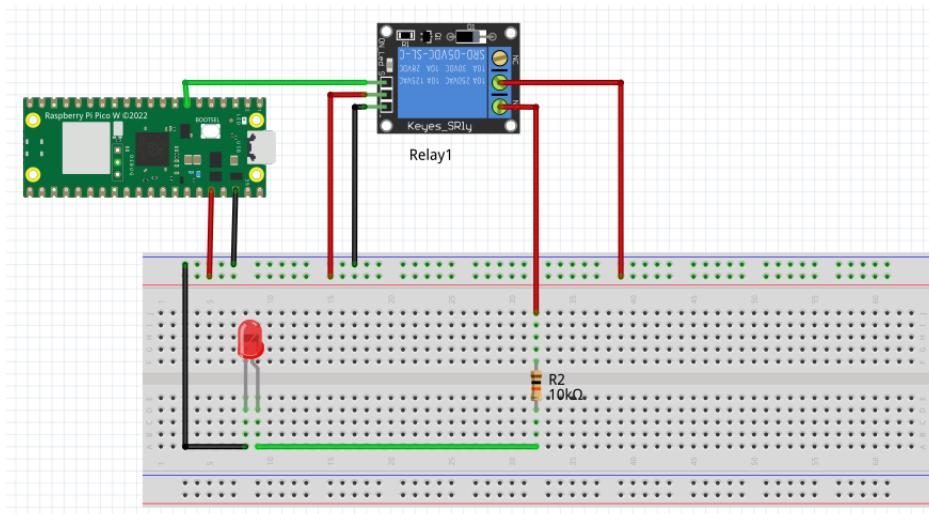


Figura C.20: Diagrama de conexiones del Actuador (II): led

Apéndice D

Documentación técnica de programación

Este manual está diseñado para que cualquier persona interesada pueda configurar la arquitectura desarrollada y poner en funcionamiento cada una de las partes que la forman.

D.1. Introducción

En este capítulo se describirán los diferentes directorios que forman el [repositorio de GitHub](#), las consideraciones a tener en cuenta a la hora de desarrollar nuevas funcionalidades, el despliegue de un entorno de pruebas y como se lleva a cabo el despliegue del sistema en un entorno de producción.

Para simplificar la gestión, desarrollo y despliegue de los diferentes servicios que forman el sistema, se han utilizado contenedores *Docker* (excepto la programación de los elementos hardware), por lo que se recomienda tener un conocimiento básico de esta tecnología.

Antes de entrar en detalle en cada uno de las carpetas relativas al software, se presenta la estructura a un más alto nivel del repositorio:

- **design:** este directorio almacena documentos relativos al diseño como son los diferentes diagramas diseñados, modelos de datos de las bases de datos o la especificación de las APIs en formato *JSON* para cargar en *Postman*.

- **docs:** este directorio está formado por este documento y la presentación final.
- **software:** este es el directorio más importante del repositorio debido a que está formado por los diferentes módulos que forman el sistema.

D.2. Estructura de directorios

Se ha decidido gestionar todas las partes del proyecto en un único repositorio *git* para simplificar la búsqueda de información; en un sistema real, posiblemente se hubiera creado un proyecto distinto para cada uno de los módulos que forma parte de la parte software. Se destacan a continuación los elementos más importantes de la carpeta *software*:

- **config:** carpeta con archivos de configuración de diferentes elementos:
 - *docker*: *dockerfiles* con las imágenes de los contenedores utilizadas.
 - *mosquitto*: formado por el archivo de configuración y el archivo con la contraseña cifrada para configurar el *broker*.
 - *nginx*: archivo de configuración del servidor web de aplicaciones.
- **web:** aplicación web del proyecto.
- **microcontrollers:** programas de los microcontroladores como el actuador y el controlador.
- **server** formado por los servicios del servidor central, entre los que se encuentran:
 - servicio de autenticación
 - servicio central con el backend
 - microservicio de recolección y almacenamiento de datos de sensores
 - microservicio de procesamiento de gestión de activación
- ***docker-compose.yml*:** definición de los servicios desplegados en *docker* utilizado por la herramienta de gestión y despliegue de multicontenedores *docker-compose*. Gracias a este archivo se pueden desplegar múltiples contenedores simultáneamente y centralizando la gestión.

- **.env_template:** plantilla con las variables de entorno utilizadas para gestionar los diferentes contenedores. La definición de este archivo es muy importante, dentro de ella se definen las variables de conexión a la base de datos, al *broker*, la clave utilizada para el cifrado de las contraseñas en el servicio de autenticación o el puerto de despliegue de los servicios y de la aplicación web, por ejemplo.

Antes de entrar en detalle respecto a la estructura de cada desarrollo software, se considera imprescindible explicar algunos aspectos del archivo *docker-compose.yml*, sin entrar en demasiados detalles. A continuación se muestra un pequeño extracto del archivo *docker-compose.yml*, mostrando un servicio basado en una imagen externa y dos servicios que hacen uso de *Dockerfiles* locales definidos por el desarrollador:

```
# Volumes
volumes:
  mysql_database_volume:
    driver: local
  mongo_database_volume:
    driver: local

# Services
services:
  broker:
    image: eclipse-mosquitto
    container_name: ${MQTT_HOST}-irrigation_broker
    restart: always
    volumes:
      - ./config/mosquitto/mosquitto.conf:/mosquitto/config/mosquitto.conf
      - ./config/mosquitto/mosquitto_passwd:/mosquitto/config/pwfile
    ports:
      - ${BROKER_PORT_BINDED}:1883:${BROKER_PORT}:1883
      - ${BROKER_WEBPORT_BINDED}:8884:${BROKER_WEBPORT}:8884
    logging:
      driver: none
    depends_on:
      - database-server

  api:
    container_name: ${API_HOST}-irrigation_api
    networks:
      default:
        aliases:
          - ${API_HOST}-irrigation_api
    build:
      context: .
      dockerfile: ./config/docker/api.Dockerfile
      args:
        - LOGS_FOLDER=logs
    restart: always
    env_file: .env
    volumes:
      - ./server/irrigation:/irrigation
      - ./logs:/irrigation/logs
      - ./env:/irrigation/.env
```

```

      - ./server/irrigation/requirements.txt:/irrigation/requirements.txt
ports:
  - ${API_PORT:-5000}:5000
depends_on:
  - database-server

web:
  container_name: irrigation_web
  networks:
    default:
      aliases:
        - irrigation_web
  build:
    context: .
    dockerfile: ./config/docker/web_server.Dockerfile
  restart: always
  env_file: .env
  ports:
    - ${WEB_PORT:-80}:80

```

- Se definen dos volúmenes para almacenar los datos de las bases de datos en un sistema de ficheros gestionados por el propio *Docker* [40].
- Todos los servicios necesitan definir un nombre identificativo y tienen que contar con una imagen base para desplegar el contenedor, puede ser un contenedor a partir de una imagen local o una imagen remota.
- El caso del contenedor del *broker* obtiene la imagen *eclipse-mosquitto* de *Docker Hub* y se especifica a través de los volúmenes internos del contenedor donde coger el archivo de configuración y el de la contraseña cifrada. Adicionalmente se puede hacer *bind* de los puertos de comunicación que abre (mediante el uso de variables de entorno).
- La diferencia con los contenedores de *api* y *web* es que se hace uso de una imagen local (véase el repositorio para más información).
- Del contenedor *api* destacar que se copia todo el código del servidor, el archivo con las variables de entorno (para configurar las comunicaciones con otros servicios), el archivo de dependencias para su instalación y se hace un *bind* de los logs.
- Como se puede apreciar, al utilizar las variables de entorno se define un valor por defecto en caso de que no esté definida en el archivo *.env* (comando *env_file: .env*).

Para finalizar, se muestra un ejemplo de un archivo *Dockerfile* para el contenedor de la *api*:

```

FROM python:3.8

LABEL author = "davalv"

ENV TZ=Europe/Madrid

ARG LOGS_FOLDER

RUN apt-get update && apt-get upgrade -y && apt-get dist-upgrade
RUN apt-get install -y \
    build-essential \
    python3 \
    python3-pip \
    && rm -rf /var/lib/apt/lists/*
RUN pip3 install --upgrade pip
RUN pip install gunicorn

RUN mkdir /irrigation
RUN mkdir /logs
RUN chmod -R 777 /irrigation
RUN chmod -R 777 /logs

VOLUME "/irrigation"
VOLUME "/logs"
VOLUME /irrigation/$LOGS_FOLDER
WORKDIR /irrigation

EXPOSE 5000

CMD pip install -r requirements.txt && gunicorn -b 0.0.0.0:5000 wsgi:app

```

Destacar que la imagen se basa en otra de *Python* y simplemente se hacen una par de cambios: nuevos volúmenes para poder copiar los ficheros, instalación de dependencias y ejecución del servicio mediante *gunicorn*.

En los siguientes subapartados se listarán los elementos más importantes a la hora de desarrollar nuevas funcionalidades o corregir bugs.

Aplicación Web (*software/web*)

- *src/components*: en este directorio se pueden encontrar los diferentes componentes web utilizados por las diferentes vistas de la aplicación, como puede ser el formulario de búsqueda o el menú superior.
- *src/views*: en este directorio se encuentra el código de las vistas como la página de inicio, la página de login y las páginas de detalle (gráfica de histórico de datos); tanto el código HTML como la gestión de llamadas a las APIs REST.
- *src/router*: en este directorio se definen las diferentes rutas de la aplicación y asociarlo a una vista determinada.

- *src/api*: en este directorio se definen los servicios de acceso al servidor central como las APIs de gestión de zonas y controladores o las APIs de histórico de sensores y activación.
- *src/services/auth*: en este directorio se definen las funciones necesarias para comprobar y asegurarse de que el usuario que entre en la aplicación dispone de un token válido.

Controlador

(*software/microcontrollers/sensor_data_controller*)

- *controller/Reader.cpp*: este archivo es el encargado de la lectura de los valores de los sensores.
- *controller/SerialManager.cpp*: este archivo es el encargado enviar los datos por conexión seria al *dispatcher*.
- *dispatcher/SerialReader.cpp*: este archivo es el encargado de la lectura de los valores de sensores por conexión serie.
- *dispatcher/FormatData.cpp*: este archivo es el encargado formatear los datos en bruto a un formato *JSON*.
- *dispatcher/config.h*: este archivo define el topic al que enviar el mensaje y los valores de conexión de red y del *broker*. **Es necesario modificarlo al cargar el programa.**

Actuador (*software/microcontrollers/actuators/relay*)

- *actuator.py*: este archivo implementa el algoritmo de gestión.
- *relay.py*: este archivo gestiona el relé.
- *mqtt.py*: este archivo implementa la gestión de recepción y envío de mensajes al *broker*.
- *secrets.py*: este archivo define los topics a los que suscribirse y emitir datos y los valores de conexión de red y del *broker*. **Es necesario modificarlo al cargar el programa.**

Servicio de autenticación (*software/server/auth*)

- *src/controller*: en este directorio se definen las APIs para gestionar usuarios y *tokens*.
- *src/dao*: en este directorio se encuentran las funciones de acceso a datos.

Servicio de riego (*software/server/irrigation*)

- *api/endpoints*: en este directorio se encuentran todos los controladores para gestionar las peticiones.
- *repositories*: en este directorio se encuentran todas las funciones de acceso a datos tanto de la base relacional (*/database*) como no relacional (*/mongo*).
- *application/actuator/models*: en este directorio se encuentran los algoritmos de activación del actuador.
- *simulators/microcontrollers*: en este directorio se encuentra un simulador del controlador para enviar datos autogenerados.
- *microservices/actuator_controller*: en este directorio se encuentra el código del microservicio encargado de la gestión de la activación del actuador.
- *microservices/data_collector*: en este directorio se encuentra el código del microservicio encargado de recogida y almacenamiento de los datos de sensores enviados por el controlador.

D.3. Manual del programador

El objetivo de este manual es servir como referencia para futuros programadores interesados en el proyecto. Se explicarán los requisitos del entorno de desarrollo, como descargar el proyecto, compilarlo, ejecutarlo y desplegarlo.

Antes de empezar a explicar la parte software, debido a que este proyecto cuenta con una pequeña parte hardware, es importante recordar que para el despliegue del proyecto es necesario disponer de un controlador y un actuador (por zona registrada) conectados con los elementos necesarios, como sensores o *relés*, siguiendo el diagrama de conexiones descrito en el

Apéndice C. Una vez se tengan listos los elementos hardware, se pasa a describir los pasos a seguir para hacer funcionar la aplicación, tanto en los microcontroladores como en el servidor central.

Entorno de desarrollo

Para que esta aplicación funcione se necesitan los siguientes elementos, programas y/o herramientas:

- Navegador web (preferiblemente *Google Chrome*).
- Git.
- *Node Version Manager*¹ para la gestión de las versiones de *Node* y *npm*.
- *Docker* y *docker-compose*.
- *VS Code* u otro IDE similar.
- IDE propio de *Arduino* (simplifica la compilación de código) con la extensión de *ESP8266* instalada².
- Placas de desarrollo: *Arduino UNO* y *Raspberry Pi pico W*³.
- Sensores y *relé*.
- Terminal (gusto personal de cada desarrollador).
- Terminal remota para conectarse a la *Raspberry Pi pico W* como puede ser *rshell*⁴.

Obtención del código fuente

- Abrir una terminal en el sistema.
- Dirigirse al directorio donde se desee copiar el repositorio.

¹Más información en el [repositorio de GitHub](#).

²Toda la información de cómo instalar el IDE y el módulo se pueden encontrar en [este post](#). Adicionalmente se indica como compilar y subir el código a la placa.

³En caso de no disponer de los microcontroladores, en el repositorio se pueden encontrar dos simuladores para el actuador y el controlador con datos, obviamente, aleatorios.

⁴Más información en el [repositorio de GitHub](#).

- Ejecutar el comando:

```
$ git clone https://github.com/davidalvarezcastro/TFM_IRRIAPP
```

- Acceder al directorio y crear una nueva rama para desarrollar una nueva funcionalidad:

```
$ git checkout -b feature
```

- Empezar a programar y, cuando se haya finalizado y pasado los *tests* definidos, crear una *Pull Requests* para ser gestionada por el *principal engineer*.

D.4. Compilación, instalación y ejecución del proyecto

En este apartado se citarán los pasos necesarios para desplegar los servicios necesarios en cada uno de los diferentes elementos que forman el sistema.

Controlador

Los pasos para compilar y subir el código al microcontrolador son los siguientes:

- Iniciar el IDE de *Arduino*.
- Pulsar *File->Open*.
- Dirigirse al directorio *software/microcontrollers/sensor_data_controller/controller* y seleccionar el archivo *.ino*.
- Seleccionamos la placa (*Board*) *Arduino UNO* y el puerto (*Port*) en el botón *Tools*.
- Pulsamos sobre la flecha del menú superior.
- Pulsar *File->Open*.
- Dirigirse al directorio *software/microcontrollers/sensor_data_controller/dispatcher* y seleccionar el archivo *.ino*.
- Seleccionamos la placa (*Board*) *ESP8266 NodeMCU 1.0* y el puerto (*Port*) en el botón *Tools*.

- Modificar el archivo *config* especificando las credenciales de la red WiFi y del *broker* y seleccionado la zona y controlador al que pertenece.
- Pulsamos sobre la flecha del menú superior y mantenemos pulsado el botón *reset* y el *flash* hasta que termine la compilación. Soltamos el botón de *flash* y cuando se visualice el porcentaje de carga, se deja de pulsar el otro botón.

Actuador

Los pasos para compilar y subir el código al microcontrolador son los siguientes:

- Abrir una terminal en el sistema y dirigirse al directorio *software/microcontrollers/actuators*.
- Enchufar la *Raspberry* por medio del USB.
- Modificar el archivo *settings* especificando las credenciales de la red WiFi y del *broker* y seleccionado la zona a la que pertenece.
- Inicializar *rshell*:

```
$ rshell
```

- Copiar todos los ficheros python a la placa:

```
$ cp actuator.py /pyboard/actuator.py
$ cp globals.py /pyboard/globals.py
$ cp main.py /pyboard/main.py
$ cp mynetwork.py /pyboard/mynetwork.py
$ cp secrets.py /pyboard/secrets.py
$ cp api.py /pyboard/api.py
$ cp mqtt.py /pyboard/mqtt.py
$ cp relay.py /pyboard/relay.py
```
- Reiniciar el microcontrolador (*unplug & plug*).

Simuladores

En caso de no contar y/o no tener acceso a los microcontroladores, el repositorio ofrece la posibilidad de simular su comportamiento con dos simuladores específicos, para el controlador y el actuador, que se pueden encontrar en formato *script* en la carpeta *software/server/irrigation/simulators/microcontrollers*: archivo *send_data.py* para el controlador y *actuator.py* para el actuador.

Los pasos para ejecutar ambos simuladores son los siguientes:

- Crear un fichero *.env* a partir de la plantilla *.env_template* especificando las ips y los puertos de los servicios y las bases de datos, el usuario administrador y su contraseña, y los *tokens* utilizados en el servicio de autenticación.

- Abrir una terminal en el sistema y dirigirse al directorio *software/server/irrigation/*.

- Crear un entorno virtual:

```
$ python3 -m venv venv
```

- Activar el entorno virtual:

```
$ source ./venv/bin/activate // ejemplo unix
```

- Instalar las dependencias en el entorno virtual (activarlo previamenet):

```
$ pip install -r requirements.txt
```

- Dirigirse al directorio *software/server/irrigation/simulators/microcontrollers*.

- Ejecutar el *script send_data.py* para simular el controlador especificando los valores de entrada:

```
$ python3 send_data.py —host 192.168.1.23 —user user
  —password password
  —summer —area 1 —controller 1
```

- Ejecutar el *script actuator.py* para simular el actuador especificando los valores de entrada:

```
$ python3 actuator.py —host 192.168.1.23 —user user
  —password password —area 1
```

- Ya se puede empezar a desarrollar nuevas funcionalidades en el sistema sin necesidad de contar con las placas de desarrollo.

Servidor Central

Los pasos para desplegar todos los servicios se muestran a continuación:

- Crear un fichero *.env* a partir de la plantilla *.env_template* especificando las ips y los puertos de los servicios y las bases de datos, el usuario administrador y su contraseña, y los *tokens* utilizados en el servicio de autenticación.

- Generar un nuevo archivo *mosquitto_passwd* en el directorio *software/config/mosquitto* con la utilidad *mosquitto_passwd* para registrar unas nuevas credenciales de acceso al servicio *MQTT* o utilizar las por defecto: *usuario - usuario*.
- Dirigirse al directorio de la aplicación web y generar otro archivo *.env* a partir de la plantilla especificando el *host* y puerto de las apis de gestión y de autenticación. Esto es debido a que se necesita compilar la aplicación web antes de desplegarla y *Vue* solo lee el archivo *.env* en el *root* del proyecto.
- Abrir una terminal en el sistema y dirigirse al directorio *software*.
- Apagar todos los contenedores:

```
$ docker-compose down
```

- Levantamos todos los contenedores. Por temas de tiempos (migraciones por parte de los servicios de riego y de autenticación), primero se levantan los contenedores de las bases de datos (*issue* detectada):

```
$ docker-compose up --build -d database-server &&
  docker-compose up --build -d
```

- Ya tenemos todos los servicios funcionando.
- Acceder a la URL <http://<ip>/irrigation/web/>.

A continuación se resume la gestión de las conexiones (gracias a las variables de entorno) en el sistema:

- En los microcontroladores la gestión se lleva a cabo mediante archivos de configuración, por lo que es necesario modificar los respectivos archivos en el controlador y el actuador, tal y como se ha descrito.
- En el caso de los servicios del servidor central, la gestión se lleva a cabo mediante un archivo *.env* central que es necesario crear en *software/.env* siguiendo la plantilla de ejemplo.
- Por último, en el caso de la aplicación web también se necesita un archivo *.env* propio debido a la nomenclatura de *Vue* y *Vite* con las variables de entorno. Esta parte se intentará optimizar de cara al futuro.

NOTA: es importante comentar que el repositorio ya cuenta con un *script* de inicialización de la base de datos que registra un par de zonas y controladores e inserta dos tipos de zonas (de monitorización y de riego) que se ejecuta al iniciar los contenedores.

Aplicación Web

En el caso de la aplicación web, es necesario diferenciar si estamos desarrollando nuevas funcionalidades o estamos desplegando la versión en producción. En el subapartado anterior, al desplegar todos los contenedores, la aplicación web está incluida. A continuación se muestran los pasos a seguir para desplegar la web en local en un entorno de desarrollo:

- Abrir una terminal en el sistema y dirigirse al directorio *software/web*.
- Crear un archivo *.env* a partir de la plantilla especificando el *host* y puerto de las APIs de gestión y de autenticación, igual que en el caso anterior.
- Se instalan las dependencias:

```
$ npm i
```

- Iniciamos aplicación (servidor de aplicaciones de desarrollo y *hot reload*):

```
$ npm run dev
```

- Acceder a la URL <http://localhost:3000/irrigation/web/>.
- En caso de querer compilar la aplicación, se ejecuta el comando *build*:

```
$ npm run build
```

- Se copia el contenido de *dist* en un servidor de aplicaciones (es lo que se hace en el *docker-compose.yml*).

D.5. Pruebas del sistema

En este apartado se describen la batería de pruebas que se han diseñado para verificar el funcionamiento de cada uno de los módulos de la aplicación y la integración de los diferentes módulos entre sí. Debido al límite de tiempo y al alcance del proyecto, se implementaron algunos tests automáticos y otros un poco más manuales:

- En el caso de los microcontroladores las pruebas se hicieron a mano y conectándose mediante puerto serie para visualizar la salida por pantalla de los *prints* dispuestos debido a la poca experiencia con este tipo de sistemas.
- Todos el código de los servicios de riego (APIs, microservicios y acceso a datos) está testeado con tests automáticos.
- La aplicación web, al ser muy sencilla, se ha probado con pruebas manuales.
- Por último, el servicio de autenticación se ha probado gracias a *Postman* con pruebas manuales. Llamando a las diferentes APIs (*login* y *logout*, que eran las que nos interesaban ahora).

Controlador

- **Lectura de datos de sensores**
 - Se conecta al Arduino y se comprueba que los valores de los sensores sean correctos.
- **Se envían los datos al *dispatcher***
 - Simulamos mediante terminal serie ser el *dispatcher* y se comprueba que se reciben los datos en el formato especificado.
- **Envío de datos**
 - Nos subscribimos al *topic* de envío de datos y se comprueba que el *broker* recibe los datos (con el formato esperado) y los reenvía a los subscriptores.

Actuador

- Recibe el evento de *open* y envía confirmación
 - Se activa el relé.
 - Nos subscribimos al *topic* de confirmación y se comprueba que el *broker* recibe la confirmación y la envía a los subscriptores.
- Recibe el evento de *open* y no envía confirmación
 - El relé ya estaba activado.
 - Nos subscribimos al *topic* de confirmación y se comprueba que el *broker* no recibe la confirmación.
- Recibe el evento de *close* y envía confirmación
 - Se desactiva el relé.
 - Nos subscribimos al *topic* de confirmación y se comprueba que el *broker* recibe la confirmación y la envía a los subscriptores.
- Recibe el evento de *close* y no envía confirmación
 - El relé ya estaba desactivado.
 - Nos subscribimos al *topic* de confirmación y se comprueba que el *broker* no recibe la confirmación.

Servicios de riego

Todos los test desarrollados en los servicios de acceso a datos de las bases de datos, en las APIs de gestión de zonas y controladores y en las APIs de históricos así como en los microservicios de recolección y procesamiento de datos se encuentran en la carpeta [software/server/irrigation/tests](#).

Para ejecutarlos es necesario seguir los siguientes pasos:

- Abrir una terminal en el sistema y dirigirse al directorio *software/server/irrigation*.
- Crear un entorno virtual:
`$ python3 -m venv venv`
- Instalar las dependencias en el entorno virtual (activarlo previamenet):

```
$ pip install -r requirements.txt
```

- Ejecutar test unitarios (generar archivos con *coverage*):

```
$ python manage.py test --unit --coverage
```

- Ejecutar test de integración (generar archivos con *coverage*):

```
$ python manage.py test --integration --coverage
```

- Los archivos se generan en */coverage*. Si se añade la opción *-open* se abrirá un navegador mostrando los resultados.

Aplicación Web

- Acceder a la aplicación sin estar identificado

- Se dirige a una ruta sin estar logueado.
- El sistema nos redirige a la pantalla de *login*.

- Credenciales incorrectas

- Se indican unas credenciales incorrectas en el formulario
- Se muestra un mensaje de error y se mantiene al usuario en la pantalla de *login*. Figura D.1.

- Credenciales correctas

- Se indican unas credenciales correctas en el formulario.
- Se redirige al usuario a la página principal.

- Logout

- Se pulsa el botón de *logout* en la parte superior.
- Se redirige al usuario a la página de *login*.

- Listado de zonas

- En la página de inicio se visualizan las zonas registradas. Figura D.2.

- Se añade una nueva zona

- Se pulsa el botón de *Add New Area*.

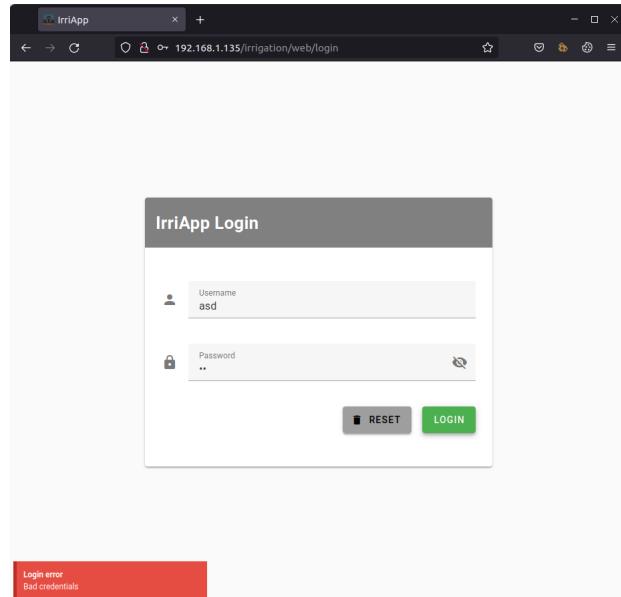


Figura D.1: Prueba web: Credenciales erróneas

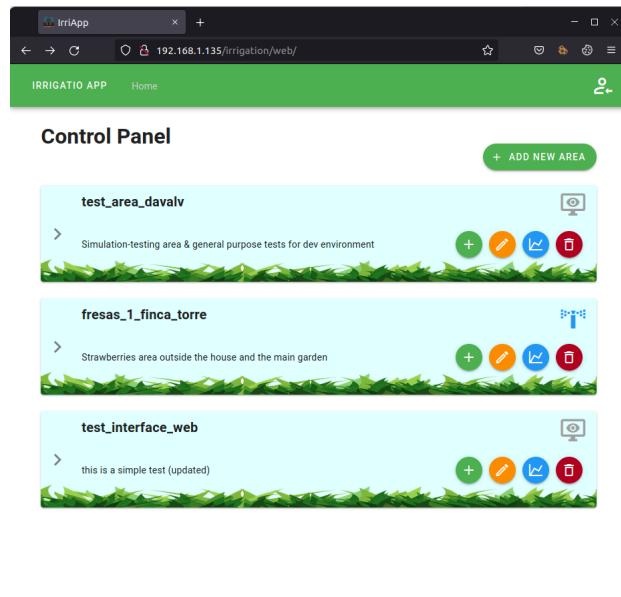


Figura D.2: Prueba web: Listado de zonas

- Se cubre el formulario y se pulsa el botón verde.
- Se cierra el modal y se muestra un mensaje conforme se ha creado la zona.
- Se actualiza el listado de zonas. Figura D.3.

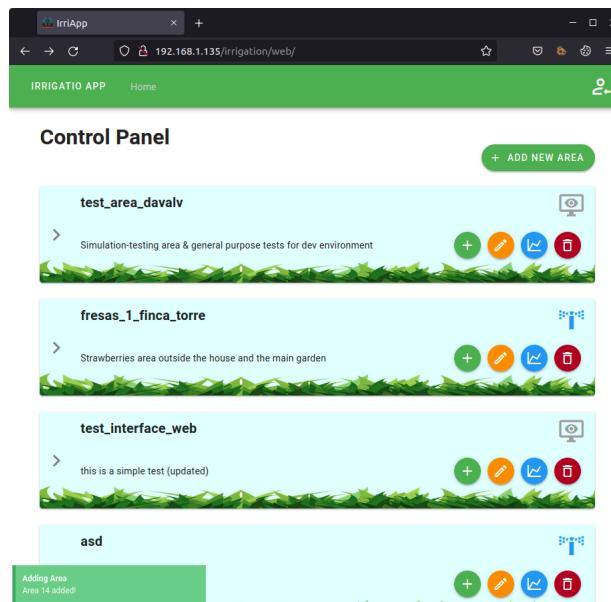


Figura D.3: Prueba web: Nueva zona

■ Se modifica una zona

- Se pulsa el botón con el icono naranja (en una zona).
- Se cubre el formulario y se pulsa el botón verde.
- Se cierra el modal y se muestra un mensaje conforme se ha actualizado la zona.
- Se actualiza el listado de zonas. Figura D.3.

■ Se elimina una zona

- Se pulsa el botón con el icono rojo (en una zona).
- Se confirma la eliminación.
- Se muestra un mensaje conforme se ha eliminado la zona.
- Se actualiza el listado de zonas. Figura D.5.

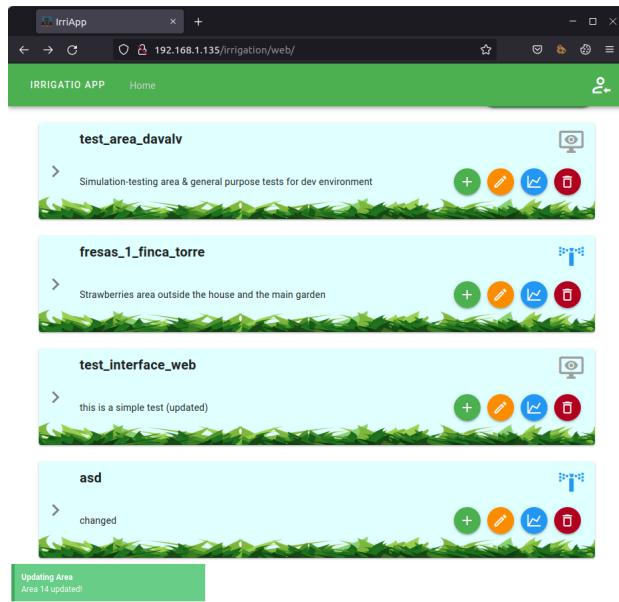


Figura D.4: Prueba web: Zona actualizada

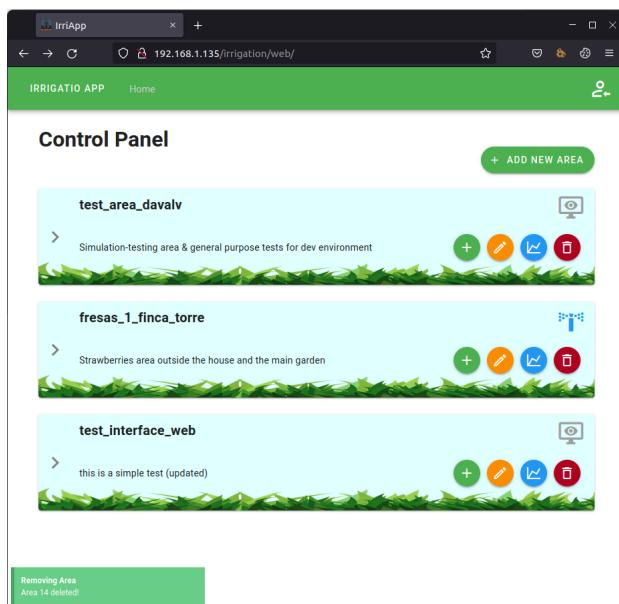


Figura D.5: Prueba web: Zona eliminada

■ Se añade un nuevo controlador

- Se pulsa el botón verde (en una zona).
- Se cubre el formulario y se pulsa el botón verde.
- Se cierra el modal y se muestra un mensaje conforme se ha creado el controlador.
- Se actualiza el listado de controladores de la zona seleccionada. Figura D.6.

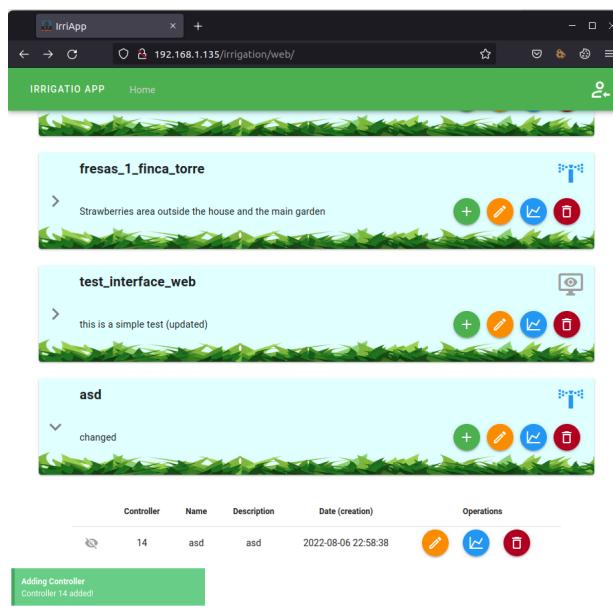


Figura D.6: Prueba web: Nuevo controlador

■ Se modifica un controlador

- Se pulsa el botón con el icono naranja (en un controlador).
- Se cubre el formulario y se pulsa el botón verde.
- Se cierra el modal y se muestra un mensaje conforme se ha actualizado el controlador.
- Se actualiza el listado de controladores de la zona seleccionada.

■ Se elimina un controlador

- Se pulsa el botón de con el icono rojo (en un controlador).
- Se confirma la eliminación.

- Se muestra un mensaje conforme se ha eliminado el controlador.
- Se actualiza el listado de controladores de la zona seleccionada.

■ **Se muestran datos de sensores de un intervalo**

- Se pulsa el botón azul (en un controlador).
- Se redirige al usuario a la vista de histórico.
- Se especifica el intervalo en el formulario de la parte superior y se pulsa el botón de búsqueda.
- Se muestra una gráfica con los datos. Figura D.7.

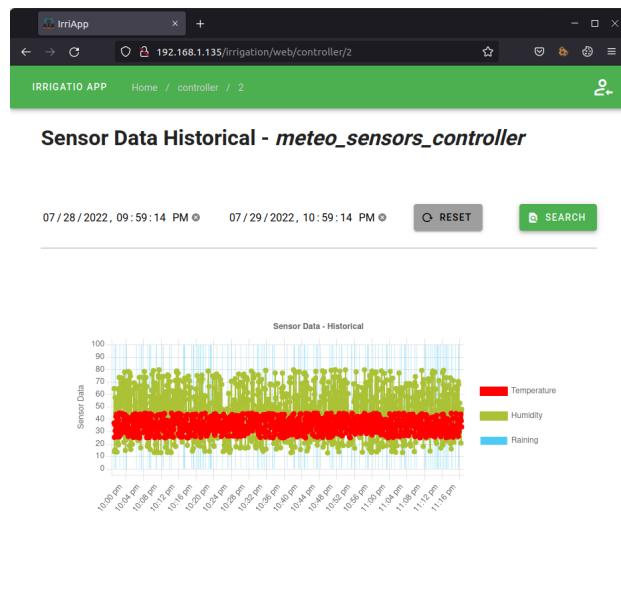


Figura D.7: Prueba web: Histórico datos sensores

■ **No se muestran datos de sensores de un intervalo**

- Se pulsa el botón azul (en un controlador).
- Se redirige al usuario a la vista de histórico.
- Se especifica el intervalo en el formulario de la parte superior y se pulsa el botón de búsqueda.
- Se muestra un mensaje conforme no hay datos en el intervalo especificado

■ **Se muestran datos de activación de un intervalo**

- Se pulsa el botón azul (en una zona).
- Se redirige al usuario a la vista de histórico.
- Se especifica el intervalo en el formulario de la parte superior y se pulsa el botón de búsqueda.
- Se muestra una gráfica con los datos. Figura D.8.

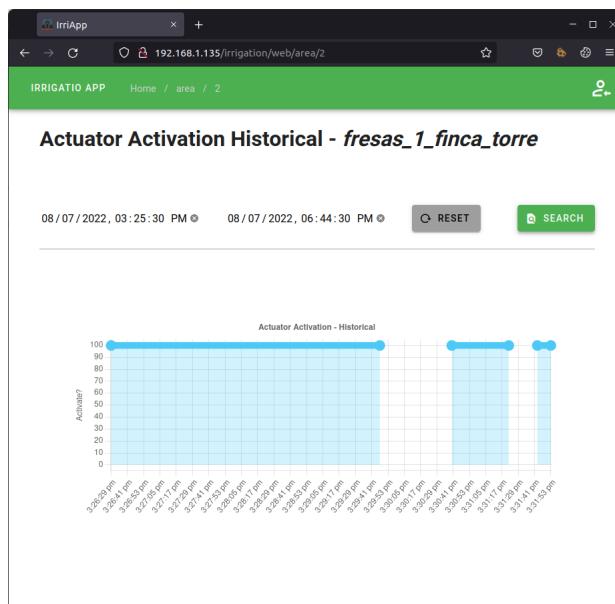


Figura D.8: Prueba web: Histórico datos activación

■ **No se muestran datos de activación de un intervalo**

- Se pulsa el botón azul (en una zona).
- Se redirige al usuario a la vista de histórico.
- Se especifica el intervalo en el formulario de la parte superior y se pulsa el botón de búsqueda.
- Se muestra un mensaje conforme no hay datos en el intervalo especificado

Apéndice E

Documentación de usuario

Este manual está diseñado para que cualquier usuario sin ninguna experiencia utilizando este sistema puede entender y ejecutar las diferentes funcionalidades ofrecidas por la aplicación web.

E.1. Introducción

En este capítulo se describirán las diferentes funcionalidades del sistema y cómo utilizar la aplicación web a los usuarios finales. Para simplificar el aprendizaje al usuario se muestra en los siguientes apartados una serie de nociones para un uso correcto de la aplicación, tomando como suposición principal que los usuarios finales nunca han utilizado esta aplicación. A mayores, también se mostrarán los pasos necesarios a realizar por parte del administrador para desplegar nuevos controladores y actuadores en el sistema.

E.2. Requisitos de usuarios

Para que el usuario final pueda ejecutar esta aplicación sin problema, son necesarios una serie de requisitos básicos:

- Navegador web (preferiblemente *Google Chrome*).
- Disponer de un servidor u otro sistema informático que tenga los servicios desplegados¹.

¹Para desplegar los servicios en una máquina en local, seguir los pasos descritos en el Apéndice D

En caso de tratarse de un usuario administrador cuyo objetivo sea registrar nuevos controladores o llevar a cabo una instalación física de alguno de los dos microcontroladores (controlador de sensores de datos o actuador), adicionalmente se tienen que cumplir los siguientes requisitos para un correcto funcionamiento del sistema:

- Para instalar un nuevo controlador (seguir planos eléctricos descritos en la Sección C.7) es necesario contar con:
 - *Arduino UNO.*
 - *NodeMCU ESP8266.*
 - Sensores.
- Para instalar un nuevo actuador (seguir planos eléctricos descritos en la Sección C.7) es necesario contar con:
 - *Raspberry Pi pico W.*
 - *Relé.*
 - Electroválvula / led.

E.3. Instalación

Al tratarse de una aplicación web, no es necesario llevar a cabo ningún tipo de instalación por parte del usuario final, siempre y cuando se disponga de un servidor con todos los servicios desplegados y en funcionamiento. La compilación y despliegue se describen en detalle en el Apéndice D.

E.4. Manual del usuario

En esta sección se detallará y explicará el funcionamiento de la aplicación web. Es importante tener en cuenta que existen dos tipos de usuarios con permisos diferentes respecto al uso de las funcionalidades de la aplicación:

- **Administrador:** puede gestionar zonas y controladores y visualizar la información de los históricos.
- **No administrador:** simplemente puede visualizar la información de los históricos.

Login

La pantalla de inicio, Figura E.1, es la de *login*, que permitirá al usuario verificar sus credenciales contra el sistema. Para acceder a cualquier otra vista de la aplicación, es necesario disponer de un *token* de acceso que se obtendrá en esta vista.

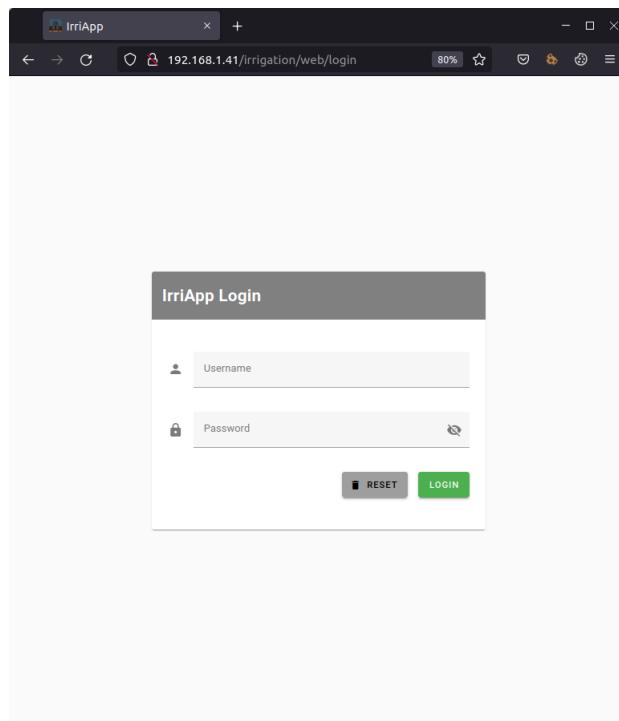


Figura E.1: Pantalla de Login

Listado de zonas

En la Figura E.2 se puede mostrar la pantalla de inicio de la aplicación, en donde el usuario puede visualizar el listado de zonas registrados, o salir de la aplicación con el botón de *logout*. En caso de que el usuario seleccione -pulse- una de las zonas se desplegará el listado de controladores asociados a cada zona.

En la pantalla inicial, el usuario administrador también puede gestionar las zonas y controladores a través de los botones asociados a cada zona/controladores.

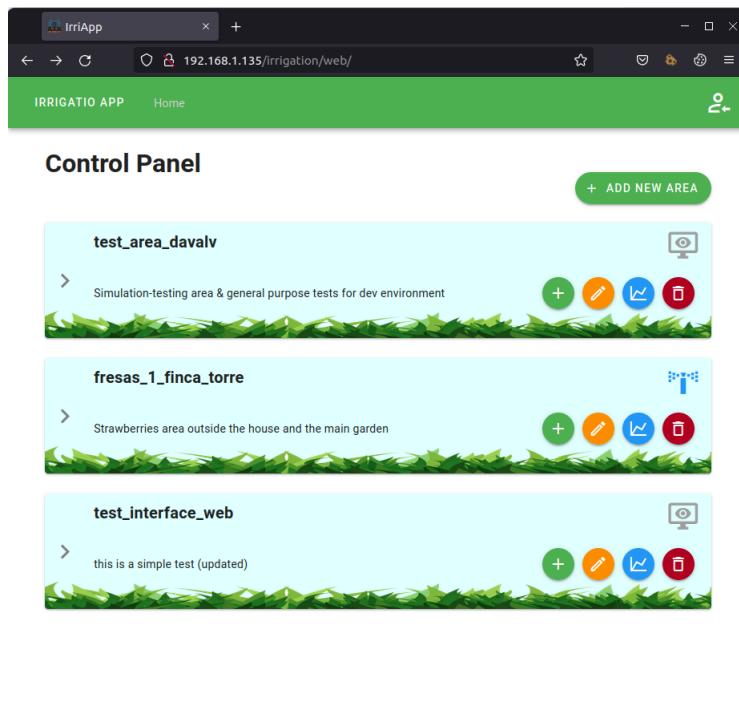


Figura E.2: Pantalla de Inicio

- En las zonas el usuario puede *añadir un nuevo controlador, modificar la zona, visualizar el histórico de activación o eliminar la zona.*
- En los controladores el usuario puede *modificar el controlador, visualizar el histórico de datos de sensores o eliminar el controlador.*

La Figura E.3 muestra la diferencia de funcionalidades que pueden llevar a cabo los dos tipos de usuarios de la aplicación.

Listado de controladores

Como se ha comentado, si el usuario pulsa sobre una de las zonas del listado se desplegará el listado de controladores asociados; tal y como se muestra en la Figura E.4.

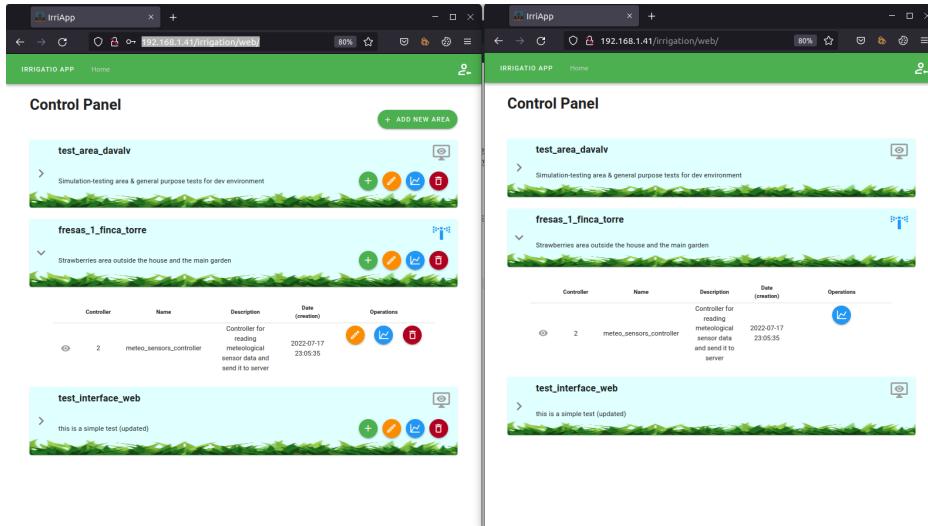


Figura E.3: Pantalla de Inicio para los diferentes usuarios

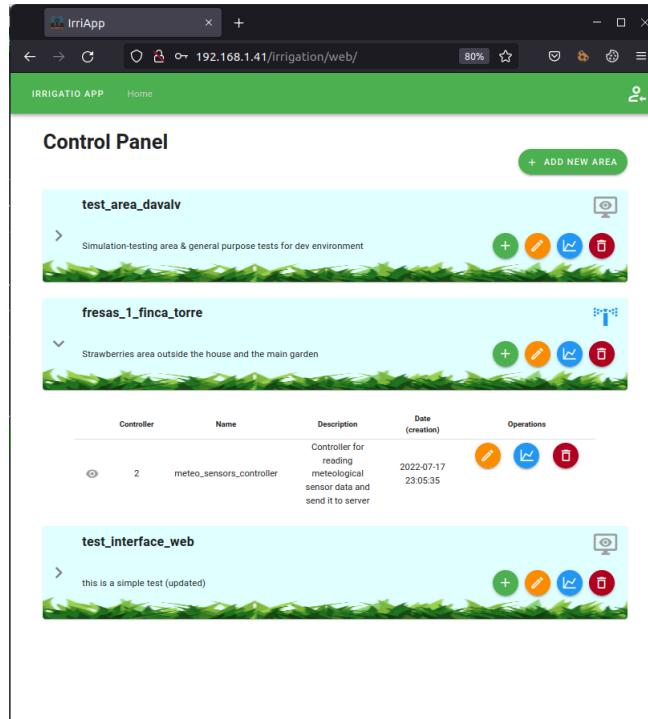


Figura E.4: Listado de controladores

Gestión de zonas (*administrador*)

En la Figura E.5 se muestran los formularios para crear o editar una zona. En el caso de querer eliminar una zona, simplemente se mostrará un diálogo de confirmación.

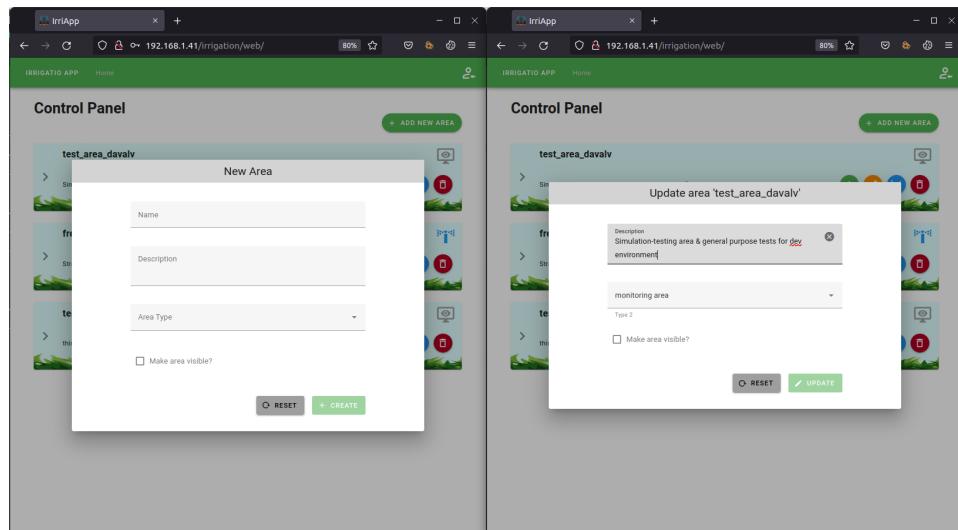


Figura E.5: Modales de gestión de zonas

Gestión de controladores (*administrador*)

En la Figura E.5 se muestran los formularios para crear o editar un controlador. En el caso de querer eliminar un controlador, simplemente se mostrará un diálogo de confirmación.

Monitorización de datos de sensores

Todos los usuarios de la aplicación pueden visualizar el histórico de datos de los sensores de un controlador. Para ello es necesario pulsar sobre la zona a la que pertenece el controlador y, a continuación, pulsar el botón azul.

Por defecto se mostrará el estado actual de los sensores, pero en el formulario superior se puede especificar un intervalo determinado y visualizar los valores tal y como se muestra en la Figura E.7.

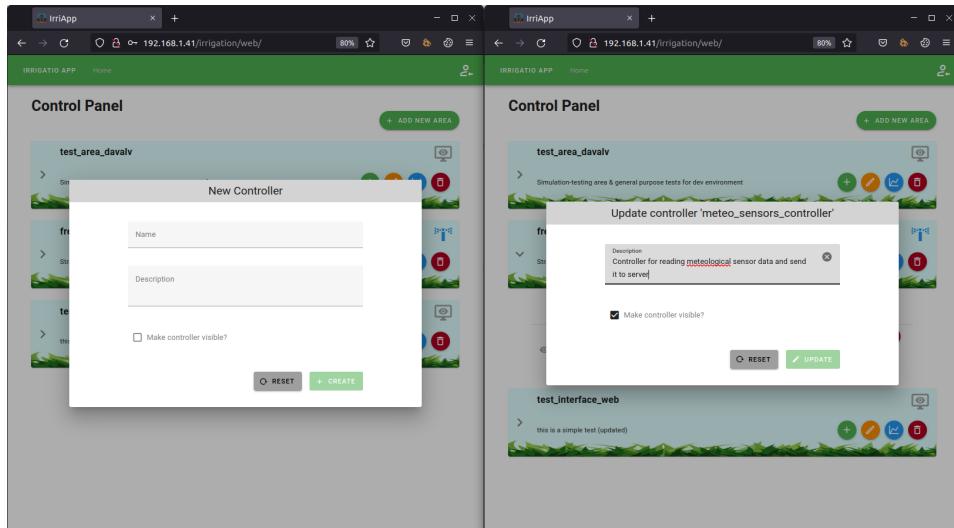


Figura E.6: Modales de gestión de controladores

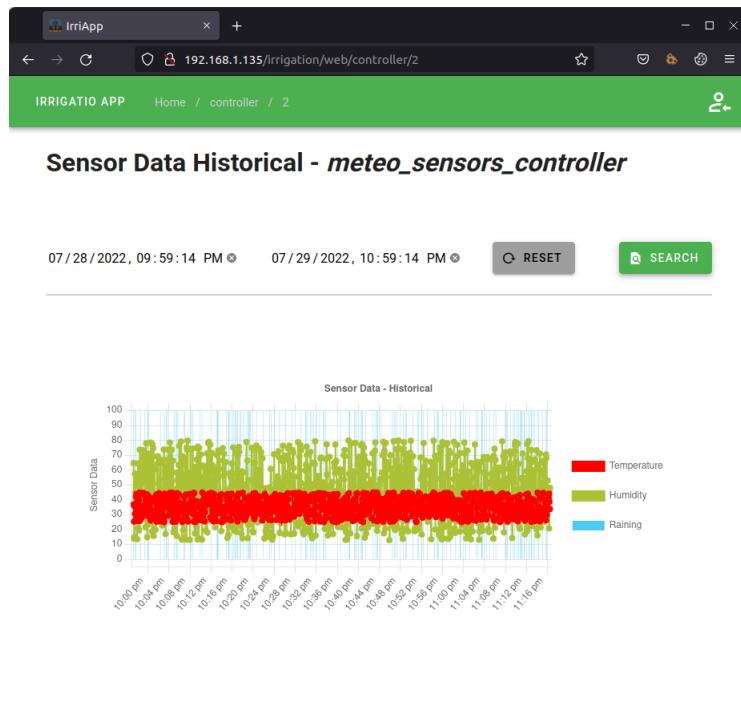


Figura E.7: Vista del histórico de datos de los sensores

Monitorización de datos de activación

Todos los usuarios de la aplicación pueden visualizar el histórico de activación de una zona. Para ello es necesario pulsar sobre el botón azul de la zona de la que se quieran visualizar los datos.

Por defecto se mostrará el estado actual del actuador, pero en el formulario superior se puede especificar un intervalo determinado y visualizar los valores tal y como se muestra en la Figura E.8.

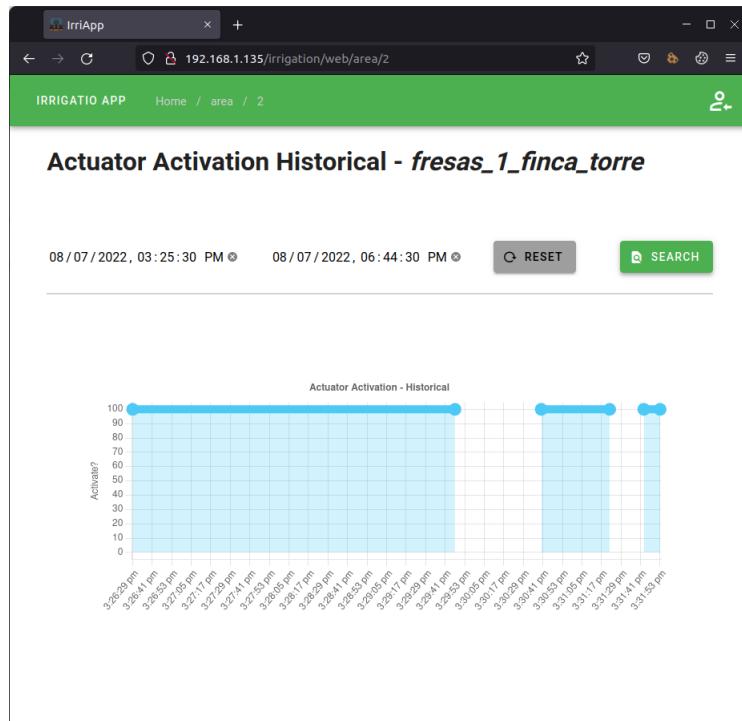


Figura E.8: Vista del histórico de activación

Logout

La aplicación gestiona el *token* de acceso de tal forma que aunque cierres el navegador, al volver a dirigirte a la URL de la aplicación, mantiene la sesión activa. En caso de querer salir de la aplicación y no ser recordado, simplemente pulsar el botón de *logout* de la parte superior de menú.

Bibliografía

- [1] Lingling Zhao, Jun Xia, Chong-Yu Xu, Zhonggen Wang, Leszek Sobkowiak, and Cangrui Long. Evapotranspiration estimation methods in hydrological models. *Journal of Geographical Sciences*, 23:359–369, 01 2013.
- [2] Mercedes Yartu, Carlos Cambra, Milagros Navarro, Carlos Rad, Ángel Arroyo, and Álvaro Herrero. Humidity forecasting in a potato plantation using time-series neural models. *Journal of Computational Science*, 59, 2022.
- [3] George H Hargreaves and Zohrab A Samani. Estimating potential evapotranspiration. *Journal of the irrigation and Drainage Division*, 108(3):225–230, 1982.
- [4] FAO. Precision irrigation. <https://www.fao.org/3/x0490e/x0490e05.htm>, (accedido 03/07/2022).
- [5] Jing Han, Haihong E, Guan Le, and Jian Du. Survey on nosql database. In *2011 6th International Conference on Pervasive Computing and Applications*, pages 363–366, 2011.
- [6] Wikipedia. Nosql — wikipedia, la enciclopedia libre. <https://en.wikipedia.org/w/index.php?title=NoSQL&oldid=836191884>, (accedido 03/07/2022).
- [7] MongoDB Docs. Databases and collections. <https://www.mongodb.com/docs/manual/core/databases-and-collections/>, (accedido 03/07/2022).

- [8] P. Félix y A. Bugarín. Conjuntos borrosos. In *Inteligencia Artificial. Técnicas, métodos y aplicaciones*, chapter 8, pages 263–306. Ed. McGraw-Hill, 2008.
- [9] Microsoft Docs. Publisher-subscriber pattern. <https://docs.microsoft.com/en-us/azure/architecture/patterns/publisher-subscriber>, (accedido 03/07/2022).
- [10] Luis Llamas. Protocolo mqtt en iot. <https://www.luisllamas.es/que-es-mqtt-su-importancia-como-protocolo-iot/>, (accedido 03/07/2022).
- [11] In Lee and Kyoochun Lee. The internet of things (iot): Applications, investments, and challenges for enterprises. *Business horizons*, 58(4):431–440, 2015.
- [12] TribaLyte Technologies. Sistemas embebidos: Conceptos fundamentales. <https://tech.tribalyte.eu/blog-sistema-embebido-caracteristicas>, (accedido 03/07/2022).
- [13] Ashok Ambardar et al. *Analog and digital signal processing*. PWS BOSTON, MA, 1995.
- [14] Yusuf Abdullahi Badamasi. The working principle of an arduino. In *2014 11th international conference on electronics, computer and computation (ICECCO)*, pages 1–4. IEEE, 2014.
- [15] Ravi Kishore Kodali and SreeRamya Soratkal. Mqtt based home automation system using esp8266. In *2016 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*, pages 1–5. IEEE, 2016.
- [16] Sean McManus and Mike Cook. *Raspberry Pi for dummies*. John Wiley & Sons, 2021.
- [17] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. Manifesto for agile software development. 2001.
- [18] José H Canós, Patricio Letelier, and M^a Carmen Penadés. Metodologías ágiles en el desarrollo de software. *Universidad Politécnica de Valencia, Valencia*, pages 1–8, 2003.
- [19] Ken Schwaber and Jeff Sutherland. The scrum guide. *Scrum Alliance*, 21(19):1, 2011.

- [20] Digité. Scrum de scrums: Un punto de partida para escalar a Ágil. <https://www.digite.com/es/agile/scrum-de-scrums/>, (accedido 03/07/2022).
- [21] David Bernstein. Containers and cloud: From lxc to docker to kubernetes. *IEEE cloud computing*, 1(3):81–84, 2014.
- [22] James Turnbull. *The Docker Book: Containerization is the new virtualization*. James Turnbull, 2014.
- [23] Docker Docs. Docker: Getting started. <https://docs.docker.com/get-started/>, (accedido 03/07/2022).
- [24] Docker Docs. Overview of docker compose. <https://docs.docker.com/compose/>, (accedido 03/07/2022).
- [25] Steve Heath. *Embedded systems design*. Elsevier, 2002.
- [26] Luis LLamas. Medir la humedad del suelo con arduino e higómetro fc-28. <https://www.luisllamas.es/arduino-humedad-suelo-fc-28/>, (accedido 03/07/2022).
- [27] Luis LLamas. Detector de lluvia con arduino y sensor fc-37. <https://www.luisllamas.es/arduino-lluvia/>, (accedido 03/07/2022).
- [28] Luis LLamas. Medir temperatura con arduino y sensor lm35. <https://www.luisllamas.es/medir-temperatura-con-arduino-y-sensor-lm35/>, (accedido 03/07/2022).
- [29] Luis LLamas. Manejar cargas de más de 220v con arduino y saliendo por relé. <https://www.luisllamas.es/arduino-salida-rele/>, (accedido 03/07/2022).
- [30] Mozilla Docs. Single-page application. <https://developer.mozilla.org/en-US/docs/Glossary/SPA>, (accedido 03/07/2022).
- [31] Gardena. Garden care system. <https://www.gardena.com/int/>, (accedido 03/07/2022).
- [32] Libelium. Libelium cloud. <https://www.libelium.com/>, (accedido 03/07/2022).
- [33] HydroPoint. The proven leader in smart water management. <https://www.hydropoint.com/>, (accedido 03/07/2022).

- [34] Instituto Tecnológico Agrario. La tecnología para optimizar tus cultivos. <https://www.sativum.es/>, (accedido 03/07/2022).
- [35] Ban Alomar and Azmi Alazzam. A smart irrigation system using iot and fuzzy logic controller. In *2018 Fifth HCT Information Technology Trends (ITT)*, pages 175–179, 2018.
- [36] Arindrajit Pal and Soumyajit Das. Development of an artificial neural network based smart irrigation system using iot, 01 2019.
- [37] Universidad de Burgos. Máster universitario en ingeniería informática: Objetivos y competencias. <https://www.ubu.es/master-universitario-en-ingenieria-informatica/informacion-basica-i/objetivos-y-competencias>, 2020 (accedido 03/07/2022).
- [38] IEEE Computer Society. Software Engineering Standards Committee and IEEE-SA Standards Board. IEEE Recommended Practice for Software Requirements Specifications. Institute of Electrical and Electronics Engineers, 1998.
- [39] Martin Fowler. *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional, 2004.
- [40] Docker Docs. Docker volumnes. <https://docs.docker.com/storage/volumes/>, (accedido 03/07/2022).