Universidad Nacional de Educación a Distancia

Escuela Técnica Superior de Ingeniería Informática

Proyecto de fin de Grado en Ingeniería Informática

Sistema de monitorización y recomendación de niveles óptimos de riego en proyectos agrícolas de gran superficie

Jordi Ferrer Beltran

Dirigido por: Emiliano Muñoz Vicente

Codirigido por: Alfonso Urquía Moraleda

Curso 2023/2024, convocatoria Septiembre

Sistema de monitorización y recomendación de niveles óptimos de riego en proyectos agrícolas de gran superficie

Proyecto de fin de Grado en Ingeniería Informática

de modalidad Externa

Realizado por: Jordi Ferrer Beltran

Dirigido por: Emiliano Muñoz Vicente

Codirigido por: Alfonso Urquía Moraleda

Fecha de lectura y defensa: Octubre 2024

Agradecimientos

A Rocío por su paciencia y apoyo.

A mi familia al completo, que se ha preocupado por mi cada día hasta que he llegado hasta aquí.

A Alfonso y Emiliano por su guía.

Resumen

En grandes proyectos agrícolas es crucial el uso eficiente de la energía y el agua empleada para irrigar los campos. Por un lado, es importante no utilizar un volumen de agua excesivo de manera que esta se filtre hasta profundidades donde deja de ser aprovechable por los cultivos y por otro lado se debe regar con la cantidad suficiente para que el agua sea fácilmente accesible para las plantas y no sea necesario realizar riegos adicionales. En este proyecto se propone el uso de transmisores iControl Soil Moisture de la empresa Proxima Systems para monitorizar la humedad del suelo y dar asistencia al usuario a la hora de planificar el próximo riego.

Se desarrolla un panel de control que permite administrar una plantación con múltiples pivotes de riego y hasta cuatro sensores en cada uno, estos sensores transmiten los datos leídos del terreno al servidor web de la aplicación mediante sockets TCP, donde se almacenan y se retransmiten a través de Websockets hacia los clientes web conectados. De esta manera, se consigue mantener el control en tiempo real del estado del suelo de la plantación. Además, se permitirá al usuario visualizar la evolución de los datos en un gráfico temporal junto a los riegos realizados para su análisis.

Finalmente, como sistema de apoyo a la decisión se implementa una herramienta que simulará el resultado del contenido de agua en cada zona de un pivote tras realizar un riego, teniendo en cuenta el volumen que aplicará en cada una. Para ello se utilizan las curvas de retención de agua según el modelo de Van Genuchten y las particularidades de cada cultivo para dar una valoración de lo adecuado que sería ese riego.

Palabras clave

Gestión agrícola, monitorización, soporte a la decisión, tiempo real, simulación, administración de pivotes de riego, humedad del suelo, riego inteligente.

Abstract

In large agricultural projects, the efficient use of energy and water for irrigating fields is crucial. On one hand, it is important not to use an excessive volume of water that seeps to depths where it becomes unavailable to the crops. On the other hand, enough water must be applied to ensure it is easily accessible to the plants, avoiding the need for additional irrigation. This project proposes the use of iControl Soil Moisture transmitters from Proxima Systems to monitor soil moisture and assist users in planning the next irrigation.

A dashboard is developed to manage a plantation with multiple irrigation pivots, each with up to four sensors. These sensors transmit the data read from the soil to the application's web server via TCP sockets, where they are stored and retransmitted via Websockets to connected web clients. This enables real-time control of soil conditions in the plantation. Additionally, users can visualize the data evolution in a time graph along with the irrigation events for analysis.

Finally, as a decision support system, a tool is implemented to simulate the water content in each zone of a pivot after irrigation, taking into account the volume applied to each zone. For this, water retention curves are used according to the Van Genuchten model and the characteristics of each crop to assess the adequacy of the irrigation.

Keywords

Agricultural management, monitoring, decision support, real-time, simulation, irrigation pivot management, soil moisture, smart irrigation.

Índice

Índice de figuras

Índice de tablas

Capítulo 1

Introducción, objetivos y estructura

1.1. Introducción

En grandes explotaciones agrícolas el uso eficiente de los recursos energéticos y del agua es esencial para su sostenibilidad y rentabilidad, ya que el derroche de estos puede causar un gran impacto negativo tanto en la empresa como en el entorno en el que se encuentra. De esta necesidad de optimización surge la motivación de este proyecto en colaboración con la empresa Proxima Systems, dedicada a soluciones de automatización y control de sistemas de riego en proyectos agrarios de gran superficie. La idea inicial es la de integrar en su aplicación los sensores iControl Soil Moisture para la monitorización de la humedad del suelo a diferentes profundidades para medir con precisión el estado del suelo y permitir al usuario tomar mejores decisiones a la hora de planificar los riegos. A esta idea se le añade también una ayuda visual que indique el límite del volumen de riego aplicable al terreno antes de causar sobre-riego.

El exceso de riego, además de provocar el desaprovechamiento de un recurso tan importante como el agua, provoca erosión en la capa superficial del suelo y la posible contaminación de almacenes de agua subterráneos debido al movimiento descendente de químicos solubles en ella, poniendo en riesgo el entorno y su explotación a largo plazo. Por otro lado, regar con poca agua también puede ser perjudicial, pues si, para cubrir la necesidad de transpiración de las plantas, se riega más veces, se producirá un consumo extra de energía para operar las bombas y los sistemas de riego y un desgaste adicional innecesario que acortará su vida útil.

Por lo tanto, se va a considerar un riego eficiente aquel que no aplica demasiada agua como para que esta se filtre a niveles de profundidad donde el cultivo ya no pueda aprovecharla y aquel que aplica suficiente como para extender al máximo el tiempo hasta el siguiente. Con este fin de optimización se ha desarrollado una herramienta de monitorización del estado de la humedad del suelo en cada pivote que permite al usuario calcular el volumen óptimo de agua aplicable en cada uno y analizar los resultados de los riegos para futuras decisiones.

Aunque el desarrollo de este proyecto es paralelo al que se realizará en la empresa para evitar la divulgación de código propietario, la base teórica y la arquitectura se ha validado de forma conjunta.

1.2. Objetivos

Con el propósito de facilitar al administrador de una plantación la configuración de los riegos de una forma eficiente, se desarrollará una aplicación web de manera que permita:

Consultar el estado de la plantación en tiempo real. El sistema recibirá los datos de humedad del suelo a tres profundidades, transmitidos por los sensores configurados, y mantendrá actualizado el panel de control del usuario. Se indicará además, con un código de colores, si el estado del suelo en ese momento supera los límites de lo considerado óptimo para el tipo de cultivo presente en ese pivote.

Dar soporte al usuario para configurar los riegos en cada pivote. Utilizando los datos de la humedad del suelo, el tipo de terreno y de cultivo, se ofrecerá al usuario una ayuda visual para elegir un volumen de agua óptimo para el riego del campo de manera que se desperdicie la menor cantidad de agua posible pero se humedezca el suelo hasta alcanzar la profundidad de las raíces del cultivo.

Analizar la evolución de la humedad del campo. Los datos de humedad se almacenarán en una base de datos y se mostrarán al usuario, cuando lo solicite, en un gráfico temporal junto al volumen aplicado en cada riego para facilitar el estudio del comportamiento del agua en cada pivote.

Como parte del proyecto se desarrollará también un simulador para reemplazar a los sensores iControl en el entorno de desarrollo y servirá también como herramienta de demostración. Esta herramienta leerá periódicamente el estado que se enviará al servidor desde un archivo de texto, donde cada línea representará un sensor diferente, identificando el número de serie y los valores de humedad del suelo a 20, 40 y 60 cm de profundidad.

Una limitación que aparece en el proyecto es el no poder utilizar la librería propietaria encargada de leer e interpretar los mensajes que provienen de los sensores iControl, para solventarlo, se desarrollará un protocolo más sencillo para la comunicación con el simulador pero teniendo en cuenta que en el futuro podría añadirse soporte al tipo de transmisores de Proxima Systems, por lo tanto, el módulo de comunicaciones deberá ser fácilmente extensible para admitir nuevos dispositivos.

Las fases de desarrollo del proyecto para llevar a cabo los objetivos mencionados serán las siguientes:

- Análisis y validación de los requisitos. Partiendo de la idea inicial para integrar y monitorizar los sensores iControl en la aplicación de Proxima Systems se establecerán las funcionalidades que debe contener el sistema. Además, ya que se trata de un proyecto en un ámbito técnico especializado, será necesario estudiar la base teórica de la propuesta y validar que se obtienen los parámetros necesarios para realizar los cálculos.

- Diseño. Una vez establecidos los requisitos funcionales y no funcionales de la tarea se definirá la arquitectura de la aplicación y las relaciones entre los diferentes actores así como las clases lógicas y servicios que componen el sistema.
- Implementación. Con el listado de tareas a realizar se procederá a su implementación una a una.
- Pruebas. Se probará que el sistema en su conjunto funciona según lo definido en la fase inicial.

1.3. Estructura

Este documento está organizado en los siguientes capítulos:

- **Capítulo 1 - Introducción:** En este capítulo se introduce el contexto del proyecto, sus objetivos y la motivación detrás del desarrollo de esta aplicación.
- **Capítulo 2 - Contexto y análisis tecnológico:** Aquí se describe el marco teórico del problema planteado y se introducen las tecnologías utilizadas en el proyecto.
- **Capítulo 3 - Análisis:** Este capítulo detalla los requisitos funcionales y no funcionales del sistema, así como los casos de uso identificados. Se definen las entidades principales que intervienen en el sistema y sus relaciones.
- **Capítulo 4 - Diseño:** En este capítulo se presenta el diseño de la aplicación, incluyendo la arquitectura del sistema, el modelo de base de datos y el protocolo de comunicación con los sensores.
- **Capítulo 5 - Implementación:** Se describe cómo se llevó a cabo la implementación de la aplicación, especificando las entidades del sistema, las operaciones CRUD, y las rutas HTTP necesarias.
- **Capítulo 6 - Pruebas:** Este capítulo contiene una evaluación de las funcionalidades implementadas, comparándolas con los casos de uso definidos previamente..
- **Capítulo 7 - Conclusiones y trabajos futuros:** En este capítulo se presentan las conclusiones del proyecto y se discuten las posibles mejoras futuras. Se reflexiona sobre la herramienta desarrollada, la metodología de trabajo y se sugieren posibles extensiones.
- **Anexo - Manual de usuario:** Se incluye un manual de usuario que describe cómo utilizar la aplicación desarrollada para gestionar los pivotes de riego y monitorizar la humedad del suelo en una plantación.

Capítulo 2

Contexto y análisis tecnológico

2.1. Introducción

En este capítulo se expondrá el marco teórico en el que se enmarcan los desafíos y objetivos del proyecto, se explicará el método de riego mediante pivotes, la medición de la humedad del suelo y como esta afecta a los cultivos, se dará contexto sobre la empresa Proxima Systems y sus soluciones y, finalmente se hará un repaso de las tecnologías que se utilizaran en el proyecto.

2.2. Irrigación mediante pivotes de riego

Los pivotes de riego son sistemas mecánicos de irrigación mediante aspersores, consisten en una tubería larga y fija en uno de sus extremos que puede girar entorno a ese punto para aplicar agua en patrón circular. El pivote suele dividirse en tramos donde cada uno tiene un par de ruedas motorizadas que permiten el movimiento de la tubería. Destacan por su capacidad de aplicar agua sobre el terreno de forma uniforme y con una eficiencia de hasta el 80 % (G. Evans 2010).

En Estados Unidos, un 29 % de todo el regadío utiliza sistemas autopropulsados de riego por aspersores, siendo la mayoría de estos de pivote central, según G. Evans (2010) . Este tipo de equipamiento ha permitido el desarrollo agrícola de terrenos poco adecuados para el riego por superficie como pueden ser suelos arenosos ligeros y arcillas con desniveles de hasta un 30 %.

Estos sistemas pueden cubrir grandes superficies y requieren muy poca operación, pueden incluso programarse de manera remota, lo que permite reducir considerablemente la mano de obra necesaria para la irrigación en grandes plantaciones (AGRIVI 2024).

En estas instalaciones, el agua se extrae de balsas, pozos u otros almacenes de agua subterránea gracias al uso de bombas, luego, en cada pivote se abre o cierra el flujo con una válvula situada en su base. Cuando se abre la válvula, el agua asciende por la tubería y la recorre hasta el final mientras va aplicándose al terreno mediante los aspersores que cuelgan en cada tramo. Si estos aspersores estuvieran configurados para expulsar agua con el mismo caudal, como aquellos que se encuentran más cerca del centro del pivote cubren un área menor que los más alejados, se regaría mucho más el cultivo en la zona interior. Para que la irrigación sea uniforme en toda la superficie deben ajustarse los aspersores para que apliquen más agua cuanto más lejos del

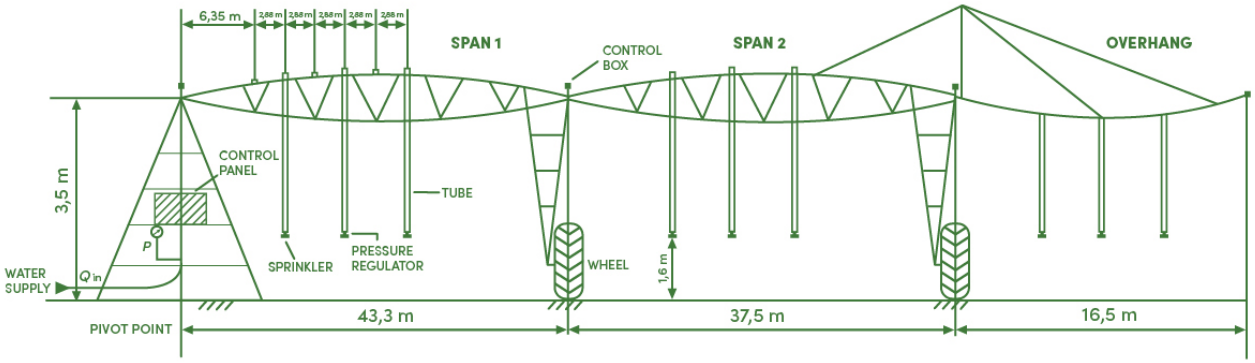

Figura 2.1: Diagrama de las partes de un pivote de riego (AGRIVI 2024).

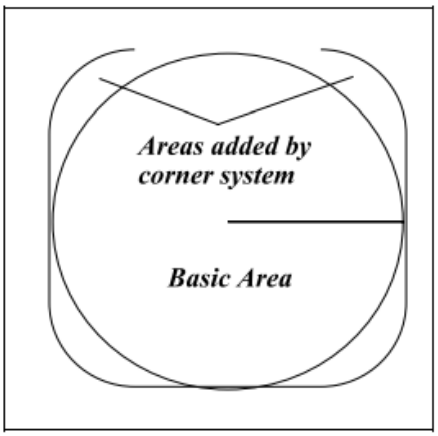

Figura 2.2: Esquema para representar el terreno adicional cubierto por los aspersores de cañón (G. Evans 2010).

centro se encuentren.

También cabe destacar, que por su naturaleza circular son menos indicados para superficies con formas rectangulares, ya que se perderá capacidad productiva en las esquinas del terreno donde no llega el agua de los aspersores que cuelgan de la tubería. Para minimizar este hecho suele incorporarse un aspersor de cañón que se activa en esas zonas y amplía el área donde puede cultivarse (Figura 2.2). Aunque son menos comunes, también existen sistemas de riego basados en aspersores que recorren el campo de manera lineal, de un extremo a otro del campo.

2.3. Proxima Systems

Proxima Systems es una empresa de Castilla y León que proporciona soluciones para habilitar el control remoto, la automatización y la monitorización de sistemas agrícolas e industriales. Su producto principal es el controlador IProx, capaz de conectarse a diferentes dispositivos utilizando entradas y salidas analógicas y digitales para transmitir su estado al panel de control del usuario y ejecutar las órdenes remotas en el sistema en el que se encuentra.

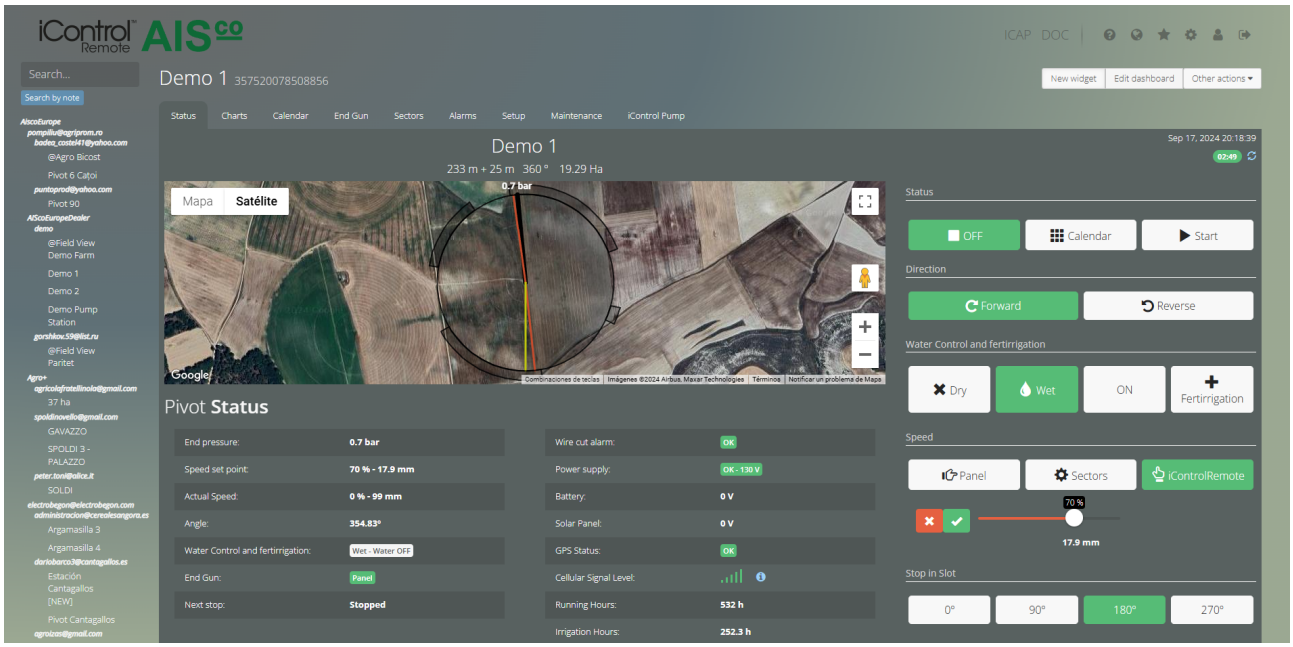

Figura 2.3: Captura de pantalla de la aplicación iControl Remote de Proxima Systems.

Aunque tiene múltiples aplicaciones, el uso principal de IProx es sobre pivotes de riego en proyectos agrícolas a gran escala, en este escenario el controlador se instala sobre la tubería del equipo y es capaz de controlar su movimiento, el paso de agua y monitorizar su estado. El usuario puede hacer estas gestiones desde la aplicación iControl Remote (Figura 2.3), en ella, además, se pueden programar riegos periódicos por sectores, configurar diferentes tipos de alarma y configurar paradas en puntos concretos.

En Proxima Systems se comercializa también el iControl Pump, una solución para controlar y optimizar las bombas de agua que trasladan el agua hasta los sistemas de riego. Esta cuenta con un algoritmo, CPO (*Cloud Pressure Optimization*), que permite reducir el consumo de energía de los riegos controlando la presión a la que funcionan las bombas de riego dependiendo del número de pivotes regando y la diferencia de altura entre ellos (Muñoz Vicente 2019).

Los transmisores iControl Soil Moisture, actualmente en desarrollo y de especial importancia en este proyecto, incluyen tres sensores de humedad y servirán como guía para el agricultor a la hora de programar el volumen de agua aplicable en los siguientes riegos.

2.4. Medición de la humedad del suelo

Es importante conocer diversos conceptos relativos a la humedad del suelo para hacer una correcta evaluación de los datos recibidos desde los transmisores iControl.

Estos transmisores contienen tres sensores capacitivos que miden el contenido de agua del suelo por su potencial matricial (SMP, por sus siglas en inglés) a tres profundidades, 20, 40 y 60 centímetros. El SMP representa la fuerza con la que las moléculas de agua se adhieren a las

partículas sólidas y entre ellas en los poros del suelo (Datta et al. 2017). Se suele denominar también succión del suelo, pues es la fuerza que deben hacer las raíces de las plantas para extraer el agua. Esta medida se expresa normalmente en kilopascales (kPa) o centibares (cb) y será siempre un valor negativo al tratarse de una presión de succión, aún así, los sensores y fuentes obvian este signo y es por eso que en la aplicación y en este documento se hace referencia al SMP sin el símbolo indicado.

El contenido de agua, además de mediante el SMP, puede medirse usando el contenido volumétrico de agua (VWC), esta medida describe el volumen de agua por unidad de volumen de suelo y se expresa como un porcentaje o un ratio. El VWC será útil para determinar el agua que se debe aplicar en un pivote para que el suelo se quede en un estado deseable y puede obtenerse a partir del SMP mediante el modelo de Van Genuchten (Van Genuchten 1980), expresado en la Ecuación 2.1:

$$\theta_{(\psi)} = \frac{\theta_s - \theta_r}{[1 + (\alpha \cdot |\psi|)^n]^{1-\frac{1}{n}}} \tag{2.1}$$

Donde:

- $\theta(\psi)$**:** Contenido volumétrico de agua por unidad de volumen de suelo L $[L^3/L^3]$.
- $|\psi|$**:** Presión de succión (cm de agua).
- θ_s**:** Contenido de agua en saturación $[L^3/L^3]$.
- θ_r**:** Contenido de agua residual $[L^3/L^3]$.
- α**:** Inversa de la presión de entrada de aire (cm^{-1}).
- $\boldsymbol{n}$**:** Medida de la distribución y tamaño de los poros del suelo.

Estos parámetros deben ser obtenidos de forma experimental para cada región o plantación, sin embargo, no son calculados de forma frecuente por los agricultores por lo que se usará una media para cada tipo de suelo según (Carsel & Parrish 1988).

2.5. Gestión del riego

Para evitar el sobre-riego, son importantes los conceptos de saturación y capacidad del suelo (FC), se dice que un suelo está saturado cuando sus poros están completamente llenos de agua, en esta situación el SMP será cercano a cero y las raíces de las plantas no podrán extraer el agua al estar drenándose por efecto de la gravedad (Allan et al. 1998). Cuando el agua de los poros

Tabla 2.1: Parámetros del modelo de Van Genuchten para diferentes texturas de suelo (Carsel & Parrish 1988)

Texture	θ_r	θ_s	α **(1/cm)**	**n**	**Ks (cm/d)**
Sand	0.045	0.43	0.145	2.68	712.80
Loamy sand	0.057	0.41	0.124	2.28	350.16
Sandy loam	0.065	0.41	0.075	1.89	106.08
Loam	0.078	0.43	0.036	1.56	24.96
Silt	0.034	0.46	0.016	1.37	6.00
Silt loam	0.067	0.45	0.020	1.41	10.80
Sandy clay loam	0.100	0.39	0.059	1.48	31.44
Clay loam	0.095	0.41	0.019	1.31	6.24
Silty clay loam	0.089	0.43	0.010	1.23	1.68
Sandy clay	0.100	0.38	0.027	1.23	2.88
Silty clay	0.070	0.36	0.005	1.09	0.48
Clay	0.068	0.38	0.008	1.09	4.80

más grandes se ha drenado se alcanza el umbral de FC, este suele considerarse como el límite superior en la gestión de los riegos (Datta et al. 2017), no es deseable que la irrigación cause que el suelo supere este límite más allá de la profundidad de las raíces pues el agua adicional percolará hacía niveles inferiores de forma que el cultivo no podrá aprovecharla. Los valores típicos de SMP para este umbral están entre los 10 y 33 kPa, dependiendo de la textura del suelo.

En el otro extremo de la saturación está el punto de marchitación permanente (PWP) que se define como el límite en el que los moléculas de agua están sujetas a las partículas del suelo con demasiada fuerza como para que las plantas puedan extraerla, esto sucede cuando el SMP alcanza valores entre 500 y 3000 kPa, aunque se considera 1500 kPa como el valor medio de SMP para el PWP en la mayoría de suelos agrícolas. Si un suelo permanece por debajo del PWP durante demasiado tiempo, el cultivo de esa zona se secará.

Se considera entonces, en (Allan et al. 1998) el total de agua accesible (TAW) como la cantidad de agua entre FC y PWP y puede calcularse con la Ecuación 2.2:

$$TAW = 1000 \cdot (\theta_{FC} - \theta_{PWP}) \cdot Z_r \tag{2.2}$$

Donde

- TAW el agua total disponible en la zona radicular [mm].

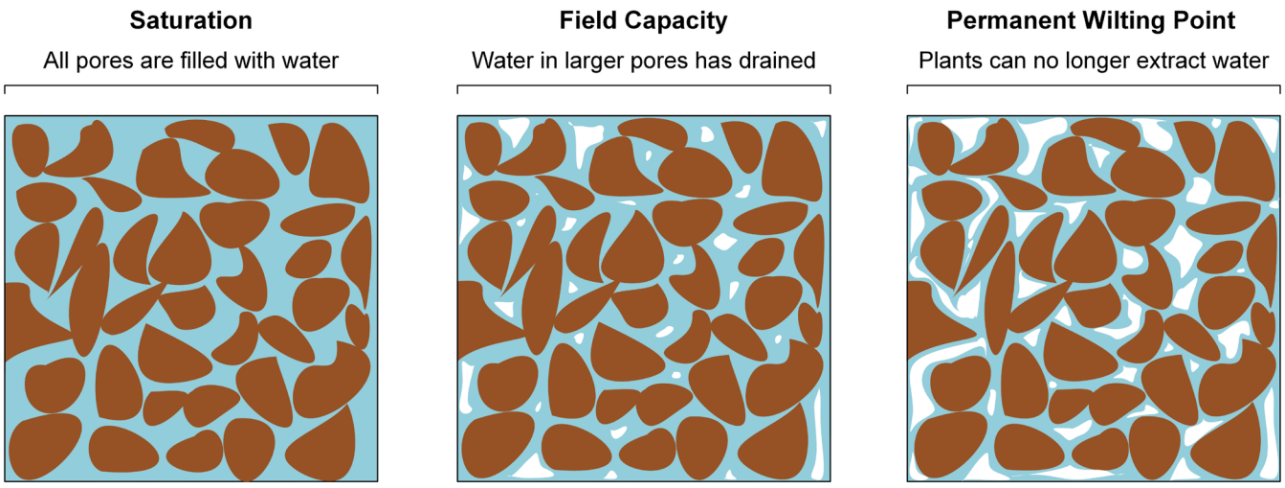

Figura 2.4: Contenido de agua en saturación, FC y PWP, (Datta et al. 2017).

- θ_{FC} el contenido de agua a FC $[m^3 m^{-3}]$.
- θ_{PWP} el contenido de agua a PWP $[m^3 m^{-3}]$.
- Z_r la profundidad de las raíces [m].

Sin embargo, los cultivos pueden empezar a sufrir estrés antes de llegar al PWP, a medida que el suelo se va secando y la fuerza necesaria para extraer el agua de los poros del suelo va aumentando, la planta deja de transportar agua suficientemente rápido como para responder a su demanda de transpiración. En (Allan et al. 1998) se define una fracción de depleción máxima del TAW de manera que permite definir un umbral a partir del cual el cultivo empezará a sufrir estrés, lo que causará un decremento de su rendimiento. Esta fracción de depleción dependerá del tipo de cultivo y de la demanda de la evapotranspiración. En la Tabla 2.2 se muestra la profundidad de las raíces y la fracción de depleción máxima para una evapotranspiración de 5mm al día.

El resultado de multiplicar TAW por esta fracción de depleción máxima es el MAD (Management Allowable Depletion), que servirá como indicador de que el cultivo comenzará a sufrir estrés al superar ese umbral de humedad en el suelo.

El objetivo del riego será por lo tanto, mantener la humedad del suelo entre FC y MAD hasta la profundidad de las raíces del cultivo.

2.6. Análisis tecnológico

En esta sección se detallan las tecnologías y componentes utilizados en el desarrollo de la aplicación, centrándose en la arquitectura cliente-servidor, los protocolos de comunicación, y el sistema de almacenamiento de datos. Estas herramientas permiten una interacción eficiente y

Tabla 2.2: Profundidad de las raíces y fracción de depleción máxima por tipo de cultivo, (Allan et al. 1998)

Cultivo	Profundidad de las raíces (m)	Fracción de depleción máxima
Brócoli	0.4-0.6	0.45
Coles de Bruselas	0.4-0.6	0.45
Repollo	0.5-0.8	0.45
Zanahorias	0.5-1.0	0.35
Coliflor	0.4-0.7	0.45
Apio	0.3-0.5	0.20
Ajo	0.3-0.5	0.30
Lechuga	0.3-0.5	0.30
Cebolla - seca	0.3-0.6	0.30
Cebolla - verde	0.3-0.6	0.30
Cebolla - semilla	0.3-0.6	0.35
Espinaca	0.3-0.5	0.20
Rábanos	0.3-0.5	0.30

en tiempo real entre los sensores de campo, el servidor, y los usuarios a través de una interfaz web.

2.6.1. Arquitectura cliente-servidor

La arquitectura cliente-servidor permite separar en dos componentes las funcionalidades de la aplicación. Por un lado el servidor se encargará del almacenamiento y procesamiento de los datos recibidos de los clientes y estos son los encargados de presentar esa información y manejar la interacción con los usuarios. En este proyecto, al tratarse de una aplicación web, el cliente será accesible desde cualquier navegador y la comunicación con el cliente se realizará utilizando el protocolo HTTP.

En este caso, el servidor realiza una función extra, que es la de recibir y procesar los datos de los sensores de humedad, para ello despliega un servidor TCP que atenderá las peticiones que lleguen por el puerto especificado, creando un socket nuevo para cada conexión. Finalmente estos datos serán retransmitidos hasta los clientes web a través de websockets. Una imagen de la arquitectura básica puede verse en la Figura 2.5.

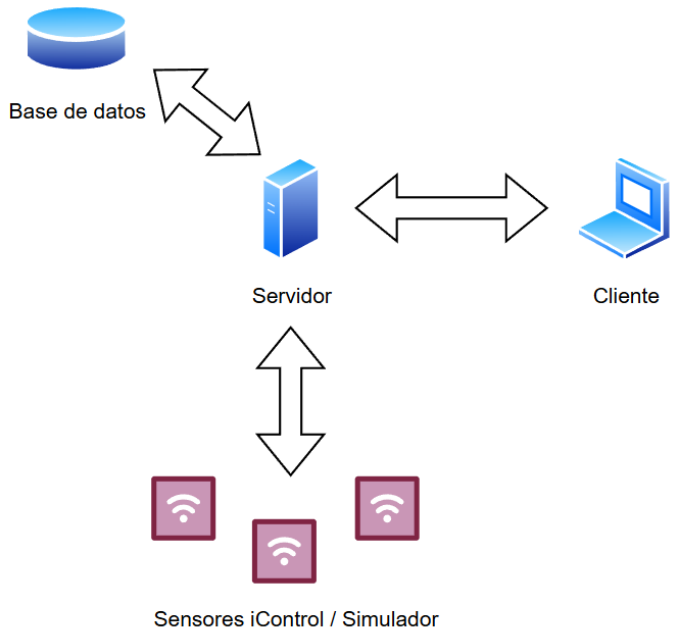

Figura 2.5: Arquitectura básica de la aplicación desarrollada.

2.6.2. Protocolo TCP

En 1974, Vint Cerf y Bob Kahn definen el Transmission Control Protocol (TCP) con el objetivo de comunicar ordenadores y terminales situados en redes diferentes y se convertirá en uno de los principales protocolos de internet complementando al Internet Protocol (IP).

TCP habilita la comunicación bidireccional (full-duplex) entre procesos definiendo en la cabecera de los mensajes el puerto destino y origen que cada uno utiliza para la conexión. Requiere que un servidor se mantenga en escucha pasiva hasta que las peticiones de los clientes establezcan una conexión. Esta conexión comienza con un acuerdo en tres pasos (*three-way handshake*) donde cliente y servidor confirman su apertura, puede verse la secuencia de mensajes que comporta ese mecanismo en la Figura 2.6, una vez la conexión se ha establecido los datos se transmiten en segmentos. Se dice entonces que TCP es un protocolo orientado a la conexión y permitirá conocer el inició y final de las comunicaciones entre los dos procesos.

Al contrario que el User Datagram Protocol (UDP), más orientado en la rapidez que en la fiabilidad de las comunicaciones, TCP implementa retransmisión de mensajes y detección de errores, lo que aumenta la robustez del protocolo pero también la latencia de los mensajes.

TCP es relevante para la aplicación desarrollada pues habilita la comunicación entre los dispositivos situados en el campo y el servidor de manera que pueda conocerse el estado de la conexión con cada uno y detectar posibles errores en la red.

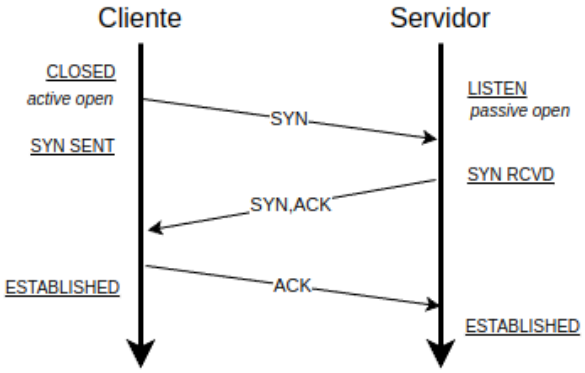

Figura 2.6: Secuencia de mensajes para establecer una conexión TCP, basado en el diagrama de estado de la conexión en (W. Eddy 2022).

2.6.3. Protocolo Websocket

El protocolo Websocket está basado en TCP y habilita la comunicación bidireccional entre aplicaciones basadas en navegador y servidores sin necesidad de abrir múltiples conexiones HTTP (Fette & Melnikov 2011).

Websocket soluciona la necesidad, en aplicaciones web como la mensajería instantánea o videojuegos, de abusar de peticiones HTTP para actualizar sus datos, causando una gran sobrecarga de la red debido al número de mensajes necesarios para abrir y cerrar cada conexión TCP y la necesidad del cliente de mantener un mapa con las conexiones entrantes para hacer el seguimiento de las respuestas.

Esta solución de utilizar una única conexión TCP para la comunicación bidireccional junto a la Websocket API, disponible en la mayoría de navegadores, es ideal para la comunicación de los datos recibidos desde los sensores de humedad del suelo hasta los clientes web de manera continua evitando la sobrecarga innecesaria de la red.

2.6.4. MongoDB

MondoDB es una sistema de base de datos noSQL donde cada entrada de una tabla se denomina documento, tiene un esquema dinámico y se almacena con una estructura de clave-valor similar a JSON.

Este tipo de bases de datos permiten la existencia de de documentos embebidos y listas que permiten reducir la necesidad de operaciones JOIN pesadas, lo que supone una ventaja frente a los sistemas relacionales. Este tipo de estructuración de los datos y el uso de índices otorgan a MongoDB un rendimiento muy alto en operaciones sobre los datos.

Además, MongoDB permite redundancia en los datos en diferentes servidores para aumentar la seguridad y la disponibilidad, también permite la escalabilidad horizontal del sistema distribuyendo fragmentos de la base de datos en múltiples clústers. Soporta también la creación de zonas de manera que las operaciones de lecturas y escrituras se dirijan únicamente a los fragmentos de las zonas que corresponden, esto, si se implementará junto a un sistema de servidor distribuido podría mejorar considerablemente el rendimiento de operaciones recibidas desde diferentes partes del mundo.

Se elige MongoDB en el servidor de la aplicación desarrollada para soportar posibles cambios en la estructura de los datos debido a, por ejemplo, cambios en los sensores de humedad y los datos que se reciben de ellos. Además es imprescindible para la escalabilidad del sistema que esos datos recibidos en tiempo real se almacenen de forma rápida para soportar el mayor número de peticiones posibles. Por último, es deseable tener la opción de escalar horizontalmente la base de datos de manera que, en el futuro, podrían crearse clústers de bases de datos en cada región donde se tengan sensores comunicando con el servidor de la aplicación y atender esas peticiones geográficamente cerca de su origen.

2.7. Conclusiones

Es este capítulo se ha expuesto el marco teórico del problema y en el que debe encuadrarse también la solución, es esencial comprender los conceptos de esta sección para definir una solución viable y útil para los usuarios finales. Además, se han comentado diversas tecnologías que se utilizarán en el proyecto y deben conocerse para comprender el desarrollo de este.

Capítulo 3

Análisis

3.1. Introducción

Como se avanzaba en el capítulo 1, el planteamiento inicial del proyecto es integrar los transmisores iControl Soil Moisture en la plataforma iControl Remote de Proxima Systems para monitorizar el estado de la humedad en el terreno de cada pivote de irrigación donde se sitúen estos sensores, así como dar un límite al volumen de irrigación teniendo en cuenta el estado del suelo para evitar el exceso de riego. Estas dos funcionalidades constituyen los requisitos funcionales principales de la aplicación desarrollada en este trabajo y han sido definidas y validadas en conjunción con Proxima Systems y *stakeholders* del producto de la empresa siguiendo métodos de análisis de ingeniería del software, como es la creación de casos de uso y el modelado de requisitos formales. Sin embargo, en este proyecto, se realiza un desarrollo paralelo a iControl Remote y será necesario añadir también la infraestructura y los prerrequisitos para soportar estas funcionalidades, simplificando y obviando muchas características de esa aplicación para poner el foco únicamente en los transmisores de humedad del suelo y la simulación de los riegos.

En este capítulo se lleva a cabo el análisis y la recopilación de requisitos del proyecto teniendo en cuenta estos aspectos, también se van a describir los casos de uso y a extraer de estos las entidades principales y como se relacionan entre ellas.

3.2. Requisitos del sistema

En esta sección se definen los requisitos funcionales y no funcionales generados a partir de la descripción de los objetivos y las necesidades del sistema, que regirán el desarrollo del sistema de monitorización y control de riego.

3.2.1. Requisitos funcionales

- **RF-01: Gestión de pivotes de riego y asignación de sensores.**
    - El sistema debe permitir al usuario crear y configurar pivotes de riego que correspondan con los pivotes existentes en el campo.

  - El usuario debe poder configurar el tipo de suelo y el cultivo que contiene el campo en el que se encuentra cada pivote.
  - El usuario debe poder asignar manualmente sensores de humedad a cada pivote de riego, indicando el número de serie de cada sensor y el ángulo, tomando el norte como 0^{o}, en el que se encuentran situados dentro del pivote.
  - El sistema debe validar que los sensores asignados a cada pivote están correctamente registrados y son únicos, evitando conflictos o duplicaciones.

- **RF-02: Consulta del estado de la plantación en tiempo real.**
  - El sistema debe recibir datos de humedad del suelo desde los sensores configurados.
  - El sistema debe actualizar continuamente el panel de control del usuario con la información recibida de los sensores.
  - El sistema debe utilizar un código de colores para indicar si el estado del suelo supera los límites considerados óptimos para el tipo de suelo y cultivo presente en cada pivote.

- **RF-03: Soporte para la configuración de riegos por sectores dinámicos.**
  - El sistema debe permitir al usuario definir sectores de riego y el volumen que aplicará en cada uno.
  - Junto a la configuración de los sectores de riego se debe mostrar una simulación que dé soporte al usuario para seleccionar el volumen óptimo de agua a aplicar en cada zona, considerando las condiciones de humedad del suelo en cada una.
  - La simulación debe actualizar la humedad del suelo en la posición de los sensores teniendo en cuenta el riego en el sector en el que se encuentran.
  - El sistema debe guardar la configuración de riego en la base de datos para que el usuario pueda evaluar el comportamiento de la humedad del suelo en base a la irrigación aplicada.

- **RF-04: Análisis de la evolución de la humedad del campo.**
  - El sistema debe almacenar los datos de humedad en una base de datos.
  - Debe permitir al usuario solicitar y visualizar gráficos temporales de la evolución de la humedad.
  - Los gráficos deben incluir el volumen de agua aplicado en cada riego, facilitando el estudio del comportamiento del agua en cada pivote.

- **RF-05: Simulador para el entorno de desarrollo.**
  - La aplicación debe incluir un simulador que sustituya a los sensores iControl en el entorno de desarrollo.
  - Este simulador debe enviar datos configurables de humedad del suelo al servidor de forma periódica.

3.2.2. Requisitos no funcionales

- **RNF-01: Accesibilidad multiplataforma.**
  - La aplicación debe ser accesible desde cualquier parte con conexión a Internet.
  - La interfaz de usuario de la aplicación debe adaptarse automáticamente a diferentes tamaños de pantalla y resoluciones.
  - Los elementos visuales, como botones, gráficos, tablas y textos, deben ser legibles tanto en dispositivos de escritorio como en dispositivos móviles.
- **RNF-03: Adaptabilidad a nuevos tipos de sensores.**
  - El sistema debe ser lo suficientemente extensible como para poder añadir nuevos protocolos de comunicación con nuevos sensores de humedad.
- **RNF-03: Usabilidad.**
  - La aplicación debe ser fácil de usar, proporcionando una interfaz intuitiva que permita a los usuarios realizar tareas comunes.
  - Debe incluir ayudas contextuales y mensajes de error claros para guiar al usuario en caso de entradas incorrectas o problemas durante el uso.
- **RNF-04: Rendimiento.**
  - Las actualizaciones de datos del estado de la humedad del suelo en el panel de control deben reflejarse en un tiempo aceptable.

3.3. Casos de uso

A continuación, se definen los casos de uso que cubren las principales funcionalidades del sistema, como la creación de pivotes de riego, la asignación de sensores, la simulación de riegos y la consulta del estado de la humedad en tiempo real. En ellos se describe la interacción entre los actores principales y el sistema, definiendo un flujo principal y varios alternativos que deben implementarse durante el desarrollo y verificarse en la fase de pruebas.

3.3.1. Añadir pivote de riego

Descripción: El administrador de la plantación añade un nuevo pivote de riego a través de un formulario en el panel de control de la aplicación.

Actor Principal: Usuario (administrador de la plantación)

Precondiciones:

- El usuario accede al panel de control de la aplicación.

Flujo Principal:

1. El usuario hace clic en el botón *"Añadir Pivote"* disponible en el panel de control.
2. El sistema muestra un formulario para la creación de un nuevo pivote de riego.
3. El usuario introduce el nombre del pivote, que debe contener únicamente letras (minúsculas o mayúsculas), espacios y números.
4. El usuario introduce el número de serie del pivote, que debe ser un código de 16 caracteres hexadecimales.
5. El usuario selecciona el tipo de suelo correspondiente al pivote de riego desde una lista desplegable.
6. El usuario selecciona el tipo de cultivo presente en el campo asociado al pivote desde otra lista desplegable.
7. El usuario revisa la información introducida y hace clic en el botón *"Guardar"*.
8. El sistema verifica que no exista otro pivote con el número de serie introducido en la base de datos.
9. Si todas las validaciones son correctas, el sistema guarda la información del nuevo pivote en la base de datos y actualiza el panel de control.
10. El sistema confirma la creación del pivote.
11. El usuario puede cancelar la operación en cualquier momento haciendo clic en el botón *"Cancelar"*.

Postcondiciones:

- El nuevo pivote de riego es creado y registrado en la base de datos del sistema.
- La información del pivote se muestra en el panel de control del usuario.

Flujo Alternativo:

3a. El nombre del pivote no cumple con el formato requerido:

- El sistema muestra un mensaje de error indicando que el nombre no es válido.
- El usuario corrige el nombre del pivote y procede con el flujo principal desde el paso 4.

4a. El número de serie del pivote no cumple con el formato requerido:

- El sistema muestra un mensaje de error.
- El usuario corrige el número de serie y procede con el flujo principal desde el paso 5.

8a. El número de serie ya existe en el sistema:

- El sistema muestra un mensaje de error indicando que ya existe un dispositivo con ese número de serie.
- El usuario introduce un nuevo número de serie y procede con el flujo principal desde el paso 4.

3.3.2. Asignar sensor a pivote de riego

Descripción: El administrador de la plantación asigna un sensor de humedad a un pivote de riego, seleccionando el ángulo dentro del pivote, verificando que el número de serie del sensor sea único y que el pivote no tenga ya 4 sensores asignados.

Actor Principal: Usuario (administrador de la plantación)

Precondiciones:

- El usuario accede al panel de control de la aplicación.
- Debe existir al menos un pivote de riego registrado en el sistema.

Flujo Principal:

1. El usuario selecciona el pivote al que desea asignar un sensor de la lista de pivotes disponibles.
2. El usuario hace clic en el botón *"Añadir Sensor"*.
3. El sistema muestra un formulario para asignar un sensor al pivote seleccionado.
4. El usuario introduce el número de serie del sensor, que debe ser un código de 16 caracteres hexadecimales.
5. El usuario selecciona el ángulo dentro del pivote (tomando el norte como 0^{o}) en el que se encuentra el sensor.
6. El usuario hace clic en el botón *"Guardar"*.
7. El sistema verifica que el número de serie del sensor no exista previamente en la base de datos.

8. El sistema verifica que el pivote seleccionado no tenga ya 4 sensores asignados.
9. Si todas las validaciones son correctas, el sistema asigna el sensor al pivote y actualiza la base de datos.
10. El sistema confirma la asignación del sensor.
11. El usuario puede cancelar la operación en cualquier momento haciendo clic en el botón *"Cancelar"*.

Postcondiciones:

- El sensor es asignado al pivote de riego seleccionado y registrado en la base de datos del sistema.
- La información actualizada del pivote se muestra en el panel de control del usuario.

Flujo Alternativo:

4a. El número de serie del sensor no cumple con el formato requerido (16 caracteres hexadecimales):

- El sistema muestra un mensaje de error indicando que el número de serie no es válido.
- El usuario corrige el número de serie del sensor y procede con el flujo principal desde el paso 4.

6a. El número de serie del sensor ya existe en la base de datos:

- El sistema muestra un mensaje de error indicando que el número de serie ya está registrado.
- El usuario introduce un nuevo número de serie y procede con el flujo principal desde el paso 4.

7a. El pivote seleccionado ya tiene 4 sensores asignados:

- El sistema muestra un mensaje de error indicando que no se pueden asignar más sensores a ese pivote.

3.3.3. Recibir datos desde un sensor y actualizar su estado en el panel de control

Descripción: El sistema recibe datos de humedad del suelo desde un sensor y actualiza automáticamente el estado de ese sensor en el panel de control del usuario.

Actores Principales:

- Sensor de humedad
- Servidor

Precondiciones:

- El sensor de humedad está configurado correctamente y conectado al sistema.
- El sensor está asignado a un pivote en la base de datos.

Flujo Principal:

1. El sensor de humedad mide los niveles de humedad del suelo.
2. El sensor envía los datos de humedad al servidor de la aplicación a través de sockets TCP indicando la profundidad a la que se ha leído el dato.
3. El servidor recibe los datos de humedad y los almacena en base de datos.
4. El servidor envía los datos recibidos al cliente a través de Websockets.
5. El cliente actualiza el estado del sensor en el panel de control del usuario, mostrando los nuevos datos de humedad con un código de colores que se corresponda con lo conveniente que son esos valores para el cultivo.

Postcondiciones:

- El estado del sensor se actualiza con los últimos datos de humedad recibidos.

Flujo Alternativo:

4a No hay ningún cliente conectado al servidor y suscrito a cambios en el sensor de humedad que envía los cambios, se termina el flujo principal tras almacenar los datos recibidos en base de datos.

3.3.4. Simular riegos y guardar la configuración utilizada

Descripción: El administrador de la plantación crea una configuración de riego para un pivote específico definiendo hasta cuatro sectores y el volumen de agua que aplicará en cada uno, cuando el usuario cambia estos valores se actualiza la previsualización del estado del campo

para mostrar los valores de humedad del suelo teóricos después de aplicar la cantidad de agua indicada en cada sector.

Actor Principal: Usuario (administrador de la plantación)

Precondiciones:

- El usuario accede al panel de control de la aplicación.

Flujo Principal:

1. El usuario hace clic en el botón *"Herramienta de riegos"* del pivote en el que desea simular el riego.
2. El sistema abre un panel que muestra una tabla con detalles del pivote seleccionado y el estado de todos los sensores asignados a ese pivote.
3. El usuario utiliza el formulario dentro del panel para añadir hasta cuatro sectores de riego, especificando sus ángulos inicial y final y el volumen de agua que aplicará en cada uno.
4. Al cambiar el valor de volumen de irrigación para un sector, el sistema actualiza automáticamente, a modo de previsualización, el resultado del contenido de agua en cada sensor dentro de ese sector utilizando las curvas de retención de agua según el modelo Van Genuchten y la cantidad de agua aplicada.
5. El usuario revisa la configuración de riego y hace clic en el botón *"Enviar"*.
6. El sistema guarda la configuración de riego en la base de datos.
7. El sistema confirma la creación exitosa de la configuración de riego.

Postcondiciones:

- La configuración de riego se guarda en la base de datos.

Flujo Alternativo:

6a. Los sectores introducidos se solapan:

- El sistema muestra un mensaje de error indicando que los sectores de riego no pueden solaparse.
- El usuario corrige los sectores y continúa con el flujo principal desde el paso 6.

Consultar la evolución de la humedad del suelo

Descripción: El usuario puede consultar la evolución de la humedad del suelo en diferentes profundidades para todos los sensores de un pivote de riego y junto al volumen de los riegos aplicados. El gráfico también muestra el rango de humedad recomendado (entre FC y MAD) para el tipo de suelo del pivote.

Actor Principal: Usuario (administrador de la plantación)

Precondiciones:

- El usuario accede al panel de control de la aplicación.
- Existe uno o más pivotes con sensores de humedad asociados.

Flujo Principal:

1. El usuario hace clic en el botón *"Evolución de los datos"* del pivote que desea analizar.
2. El sistema carga de forma paginada los datos de humedad del primer sensor del pivote seleccionado.
3. En cuanto los primeros datos están disponibles, el sistema muestra un gráfico con los datos de humedad de los últimos treinta días en las tres profundidades disponibles (20, 40, y 60 cm). El gráfico incluye los volúmenes de riego aplicados por el usuario en el mismo período. Se visualiza el rango recomendado de humedad (entre FC y MAD) como una banda verde en el gráfico, indicando el rango de humedad óptima para el tipo de suelo del pivote.
4. El usuario visualiza datos de todos sensores del mismo pivote utilizando los botones de navegación dentro del gráfico. Al seleccionar otro sensor, el gráfico se actualiza automáticamente para mostrar los datos de humedad correspondientes a ese dispositivo.
5. El usuario selecciona un rango de fechas dentro del periodo para el que se tienen datos en ese sensor. Al cambiar el rango de fechas, el gráfico se actualiza para reflejar los datos correspondientes al nuevo intervalo de tiempo.

Postcondiciones:

- El usuario visualiza un gráfico actualizado que muestra la evolución de la humedad del suelo, los volúmenes de riego aplicados, y el rango recomendado de humedad para el tipo de suelo del pivote seleccionado.

Flujo Alternativo

2a No hay datos de humedad del suelo para el sensor seleccionado:

- Se muestra un mensaje indicando este hecho.

3.4. Entidades principales

A partir de los requisitos funcionales y la descripción de los casos de uso de la aplicación se extraen las siguientes entidades y sus respectivos atributos, representadas en la Figura 3.1:

- **Pivote**. Representa la máquina encargada de regar el campo y se identifica según el número de serie del iControl que monitoriza su estado y lo controla de forma remota.
  - **Nombre**. Permite identificar de forma sencilla cada equipo.
  - **Número de serie**. Identifica de forma única cada pivote relacionándolo con una máquina real en el campo.
- **Sensor**. Representa un dispositivo de medición de la humedad del suelo, contendrá múltiples sensores que miden el contenido de agua a diferentes profundidades y las enviarán al servidor.
  - **Número de serie**. Identifica el transmisor de manera única.
  - **Ángulo**. Indica en que posición dentro del pivote se ha instalado el sensor. Se toma el norte magnético como grado 0 e incrementando en sentido horario hasta 360 en la circunferencia que dibuja el pivote al recorrer el campo.
- **DatoHumedadSuelo**. Representa un dato generado por un sensor de humedad del suelo en un momento dado.
  - **Fecha**. Fecha de la lectura.
  - **Profundidad**. Profundidad a la que se ha leído el dato.
  - **Valor**. Valor de la humedad del suelo en la profundidad indicada.
- **Cultivo**. El tipo de cultivo presente en el campo irrigado por un pivote.
  - **Nombre**. Nombre del cultivo.
  - **Profundidad de las raíces**. Indica la profundidad máxima de las raíces según (Allan et al. 1998).
  - **Depleción máxima**. Fracción de la disminución del TAW que permite el cultivo antes de sufrir estrés. (Allan et al. 1998)
- **Riego**. Configuración del riego creada por el usuario.
  - **Fecha**. Fecha en la que se crea el riego.
  - **Ángulo inicial**. Ángulo inicial del arco que realizará el pivote al regar.
  - **Ángulo final**. Ángulo final del arco.

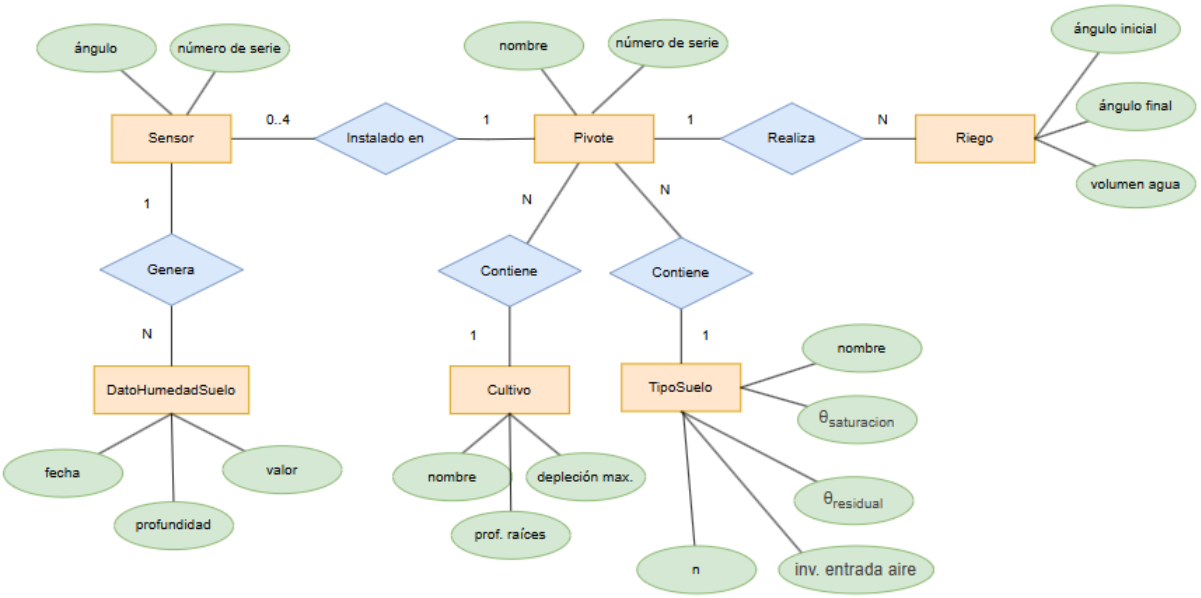

Figura 3.1: Diagrama Entidad-Relación.

- **Volumen de agua**. Cantidad de agua en mm aplicada por el pivote.

- **TipoSuelo**. Describe el tipo de suelo presente en el campo donde se encuentra el pivote, contiene los parámetros del modelo Van Genuchten para calcular la curva de retención de agua de la tabla 2.1.

  - **Nombre**. Identifica la textura del suelo.
  - **Contenido de agua en saturación**. Parámetro θ_s, indica el contenido de agua cuando el suelo está saturado.
  - **Contenido de agua residual**. Parámetro θ_r, indica el contenido de agua en el rango PWP.
  - **Inversa de entrada de aire**. Parámetro α del modelo Van Genuchten.
  - **n**. Parámetro n, es una medida de la distribución y el tamaño de los poros del suelo.

3.5. Conclusiones

En este capítulo se ha llevado a cabo un análisis de los requisitos del sistema con el objetivo de garantizar que la aplicación propuesta cumpla con los objetivos establecidos. Los requisitos funcionales abarcan la gestión de los pivotes de riego, la asignación de sensores, la recepción de datos en tiempo real y la configuración dinámica de sectores de riego. Adicionalmente, se ha considerado un simulador para el entorno de desarrollo, permitiendo la validación del sistema sin necesidad de depender exclusivamente de sensores físicos.

Se han definido casos de uso que cubren las interacciones entre el usuario y el sistema: la creación y administración de pivotes, la asignación de sensores y la simulación de riegos; y el proceso de colección de datos desde los sensores.

Finalmente, se han identificado las principales entidades del sistema y sus relaciones, permitiendo una estructura bien organizada y eficiente para la manipulación de los datos. Este análisis establece una estructura sólida para la implementación, asegurando que las funciones del sistema aborden las necesidades del proyecto.

Capítulo 4

Diseño

4.1. Introducción

En este capítulo se detallará el diseño del sistema de manera que permita cumplir con los requisitos funcionales y no funcionales definidos en el Capítulo 3.

Primero se definirá una arquitectura cliente-servidor que habilite la interacción entre los sensores instalados en el campo y el panel de control del usuario. También se describirá el esquema de la base de datos adaptando el diagrama de las entidades principales de la aplicación de la Figura 3.1 a colecciones de MongoDB.

Finalmente, se detallará el funcionamiento del simulador de los sensores de humedad del suelo para sustituir a los dispositivos reales en el entorno de desarrollo.

4.2. Arquitectura

La aplicación debe ser accesible desde cualquier parte con conexión a internet, como se documenta en el requisito no funcional RNF-01, para que el administrador de la plantación pueda, por ejemplo, realizar consultas del estado del suelo desde su teléfono cuando se encuentra en el terreno y realizar un análisis de la evolución de los datos según los últimos riegos desde un ordenador de sobremesa cuando está en la oficina. Para cumplir con este requisito se decide desarrollar una aplicación web con la arquitectura de la Figura 4.1.

En el servidor se desplegarán tanto la aplicación backend como frontend. La primera se encargará de crear el servidor de sockets TCP al que se conectarán los diferentes sensores de la plantación y de cargar y almacenar todos los datos que reciba de ellos, también es responsable de responder a las peticiones HTTP del cliente y realizar las operaciones CRUD sobre las entidades de la aplicación y a su vez mantener el servicio de Websockets para la retransmisión en tiempo real de los datos leídos por los sensores. Por otro lado, la aplicación frontend se hará accesible por el puerto 80 del servidor y será la encargada de enviar al cliente web los archivos necesarios para cargar el panel de control, desde ahí el usuario podrá realizar todas las funciones definidas los casos de uso.

Como base de datos se decide utilizar MongoDB para aligerar la inserción de los datos en

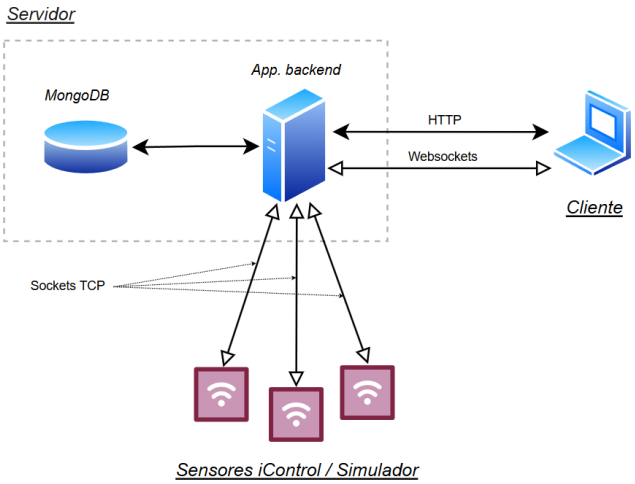

Figura 4.1: Arquitectura de la aplicación.

tiempo real y por la flexibilidad de los esquemas de sus documentos, necesaria si se deciden añadir nuevos tipos de sensores o nuevos protocolos que requieran atributos diferentes a los definidos. Esta base de datos se instala inicialmente en el mismo equipo que la aplicación web para simplificar la arquitectura y facilitar el mantenimiento, sin embargo, si se necesitara escalar la capacidad de almacenamiento esta podría migrarse fácilmente a otro servidor haciendo una copia de los datos y cambiando la cadena de conexión en el backend para apuntar a la nueva localización.

4.3. Modelo de la base de datos

A partir de las entidades descritas en la Sección 3.4 se ha creado el modelo de la base de datos como se muestra en la Figura 4.2. Cada clase de este modelo no tiene por qué corresponder con una tabla en base de datos, ya que se hace uso de los documentos embebidos de MongoDB, es el caso de los sensores y los sectores de riego, que serán atributos de tipo array dentro de *Pivot* e *Irrigation*, respectivamente.

Además, es necesario destacar las tablas *LastState* y *ValueChange*, que almacenan los datos recibidos desde los sensores y ambas se corresponden con la entidad *DatoHumedadSuelo* de la figura 3.1. Como su propio nombre indica *LastState* únicamente almacenará el último valor recibido para cada atributo y cada sensor mientras que *ValueChange* contiene toda la información recibida hasta el momento. Aunque ambas tablas podrían combinarse, se ha dividirlas de esta manera para facilitar y aligerar la consulta de los últimos datos recibidos.

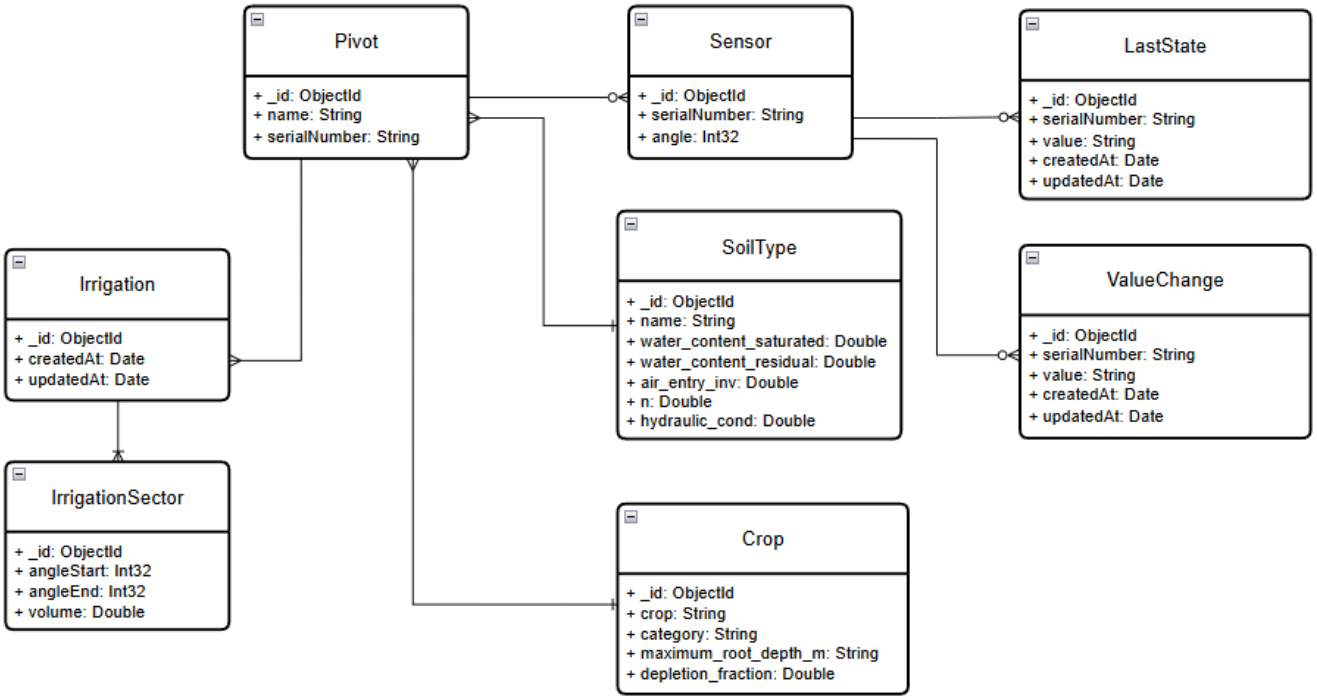

Figura 4.2: Esquema de la base de datos.

simulator > sensor-status.txt

```
1   987456215654=43,50,150
2   123456789456=70,80,120
3   786542587854=180, 150, 200
4   123456789458=180, 150, 200
```

Figura 4.3: Ejemplo de los estados de diferentes sensores simulados.

4.4. Protocolo de comunicación y funcionamiento del simulador

Para evitar la publicación de código propietario como es la librería encargada de deserializar los mensaje codificados de los sensores iControl Soil Moisture, es necesario implementar un simulador y especificar un protocolo de mensajes para reproducir el comportamiento de estos transmisores. Esta herramienta debe permitir al desarrollador cambiar los valores de la humedad del suelo y ver estos cambios reflejados en tiempo real en el servidor.

Para conseguir este efecto, se definirá en un archivo los números de serie de los sensores que se quieren simular, y tras crear una conexión TCP con el servidor para cada uno, el simulador transmitirá los datos que se definen en un segundo fichero con el formato de la Figura 4.3. En él, cada línea representa el estado de un sensor, el simulador debe ser capaz de detectar cambios en este fichero y transmitir los nuevos datos al servidor cuando se realicen.

La comunicación desde el simulador utiliza una versión simplificada del protocolo que utilizan los sensores iControl Soil Moisture, en este caso se utilizarán mensajes con la siguiente forma: *#TIPO_DE_MENSAJE:DATOS*, los tipos de mensaje y los datos que contienen pueden verse en la Tabla 4.1.

Cada sensor empezará la comunicación con un mensaje de identificación *SENSOR_CONNECTION* con su número de serie, este se utilizará en el servidor para asociar cada socket conectado con el dispositivo correspondiente. Tras este paso, enviará el estado según se indica en su línea del fichero correspondiente con un mensaje de tipo *DATA*. Este último tipo de mensaje se volverá a enviar siempre que se modifique el archivo con los estados de los sensores.

Tabla 4.1: Tipos de mensaje en la comunicación con los sensores.

Tipo de mensaje	**Datos**
SENSOR_CONNECTION	Número de serie
DATA	sen1=VALOR1,sen2=VALOR2,sen3=VALOR3

Los datos *sen1, sen2 y sen3* enviados en los mensajes de tipo *DATA* se corresponden con el PMS a 20, 40 y 60 cm de profundidad en un sensor y serán siempre valores enteros entre 0 y 240 kPa.

4.5. Conclusiones

Teniendo en cuenta las necesidades del sistema de mantener un panel de control actualizado con datos en tiempo real, se ha diseñado una arquitectura cliente-servidor con comunicaciones mediante sockets TCP entre el servidor y los sensores y a través de Websockets con los clientes web. Además se ha decidido utilizar MongoDB como base de datos para soportar cierta flexibilidad en los esquemas de bases de datos en caso de que se decidan añadir nuevos tipos de dispositivos IoT y estos requieran nuevos atributos. También, se ha modelado el esquema de la base de datos partiendo de las entidades definidas en el Capítulo 3 haciendo uso de documentos embebidos para facilitar la optimización de las consultas y operaciones sobre los datos.

Por último, se ha descrito el funcionamiento del simulador de los sensores de humedad para que permita al desarrollador enviar datos al servidor de forma similar a como lo hacen los dispositivos instalados en el campo pero utilizando un protocolo simplificado.

Capítulo 5

Implementación

5.1. Introducción

El desarrollo de la aplicación web se ha realizando utilizando dos frameworks de desarrollo de Typescript: NestJS en el backend y Angular en el frontend. La decisión para usar estas tecnologías responde principalmente a la familiaridad del desarrollador con ellas y la ventaja de utilizar el mismo lenguaje en ambos lados de la aplicación, que hace que la transición de trabajar en uno a otro sea más sencilla. Es cierto que se sacrifica algo de rendimiento al utilizar un lenguaje interpretado como Javascript en el servidor y puede traducirse en un decremento de las peticiones que puede atender de forma simultánea, pero ya que no es una aplicación que se dirija al público masivo si no que más bien sería utilizada por las personas responsables de administrar una plantación concreta, no es un aspecto crítico a tener en cuenta.

También se ha desarrollado un simulador de sensores de humedad en NodeJS para replicar el comportamiento de los sensores iControl Soil Moisture siguiendo la definición que se hace en la Sección 4.4.

En este capítulo se describirá primero como se implementan las entidades principales de la aplicación y las operaciones que pueden realizarse sobre ellas, luego se dedicará una sección para explicar el desarrollo de cada uno de los requisitos funcionales descritos en el Apartado 3.3.

5.2. Definición de las entidades principales, operaciones CRUD y rutas HTTP

En la carpeta raíz del proyecto del backend se define un módulo para cada entidad principal de la aplicación, estos módulos contienen los esquemas, servicios y controladores necesarios para modelar los datos, realizar operaciones CRUD sobre ellas y definir las rutas de la api.

5.2.1. Esquemas

Se ha utilizado *Mongoose* para modelar los datos de la aplicación, esta librería permite la definición de esquemas que describen los atributos de los documentos de cada colección y sirven para generar los modelos de cada entidad. Estos objetos son los encargados de crear los documentos que se almacenarán finalmente en base de datos.

En el Listado 2.1 se incluye el código utilizado para generar el esquema de la entidad *Pivot*, tal y como se describe en el diagrama de la Figura 4.2. En este ejemplo se ilustra también el uso de documentos embebidos en el atributo *sensors*. El esquema de la clase *Sensor* se definirá de forma similar en un archivo diferente y generará su propio modelo, pero serán almacenados siempre como atributos de los pivotes. Como se ha comentado en el apartado correspondiente de estado del arte, esto permite evitar el uso de operaciones JOIN y optimiza la carga de los datos.

```
@Schema()
export class Pivot {
  @Prop({ required: true })
  name: string;

  @Prop({ required: true })
  serialNumber: string;

  @Prop()
  sensors: Sensor[] = [];

  @Prop({ type: mongoose.Schema.Types.ObjectId, ref: 'SoilType' })
  soilType: SoilType;

  @Prop({ type: mongoose.Schema.Types.ObjectId, ref: 'Crop'})
  crop: Crop;
}

export const PivotSchema = SchemaFactory.createForClass(Pivot);
```

Listado 5.1: Esquema para la entidad Pivot.

Se creará por tanto un esquema por cada clase del modelo de datos en su módulo correspondiente para definir todas las colecciones de la base de datos y generar sus modelos.

5.2.2. Controladores

Los controladores definen las rutas de la API que el servidor pone a disposición de los clientes para realizar operaciones CRUD sobre las entidades del proyecto. Se utiliza el decorador *@Controller(NOMBRE_ENTIDAD)* de NestJS para definir un controlador y el prefijo de las rutas para esa clase. Dentro del manejador de cada ruta se harán las comprobaciones pertinentes para

asegurar que los parámetros de entrada son correctos y se devuelve el resultado de la operación realizada.

```
@Controller('pivots')
export class PivotsController {
  constructor(private pivotsService: PivotsService) {}

  @Post()
  async create(
    @Body() createPivotDto: CreatePivotDto,
  ): Promise<Pivot | OperationError> {
    const pivotExists = await this.pivotsService.pivotExists(
      createPivotDto.serialNumber,
    );

    if (pivotExists) {
      return {
        error: `Pivot ${createPivotDto.serialNumber} already exist.`,
      };
    }

    return this.pivotsService.create(createPivotDto);
  }

  @Patch()
  async update(
    @Body() updatePivotInput: UpdatePivotDto,
  ): Promise<Pivot | OperationError> {
    const pivotExists = await this.pivotsService.pivotExists(
      updatePivotInput.serialNumber,
    );

    if (!pivotExists) {
      return {
        error: `Pivot ${updatePivotInput.serialNumber} does not exist
    `,
      };
    }

    return this.pivotsService.update(updatePivotInput);
  }

  .
  .
  .
}
```

Listado 5.2: Código parcial del controlador de la entidad Pivot.

Se utilizan también las palabras clave de Mongoose *@GET, @PATH* y *@POST* para definir el método HTTP que debe utilizarse en cada ruta de manera que este se corresponda con el tipo de operación que realiza siguiendo la definición de una API REST. Es decir, se utiliza el método GET cuando se desean obtener datos, POST cuando quiere se quiere almacenar nueva

información y PATCH cuando se desea modificar algún dato dentro de un documento existente.

5.2.3. Servicios

Los servicios contienen la lógica para acceder a base de datos y se utilizan en los controladores por inyección de dependencias de NestJS, un ejemplo de esto puede verse en la línea 3 de la Figura 5.2. Separar los servicios de los controladores es una manera de encapsular la lógica de acceso a base de datos y la definición de las rutas en sus clases respectivas de manera que cada una termine siendo más sencilla y sea responsable únicamente de una tarea.

En los servicios se utilizan los modelos generados para cada tipo de entidad para crear y editar documentos en cada colección. En el Ejemplo 5.3, se muestra un fragmento de la clase *PivotsService*, en su constructor se inyectan los modelos de los que depende la entidad *Pivot*, luego, en la función *create* se utiliza *pivotModel* para crear un documento a partir de los datos recibidos y almacenarlo en base de datos. En el método *update* se realiza una actualización de un documento existente y se devuelve el resultado, es necesario hacer uso de los modelos de los esquemas que se referencian dentro de *Pivot* para que en el resultado se incluyan también los documentos correspondientes.

```
1  @Injectable()
2  export class PivotsService {
3    constructor(
4      @InjectModel('pivots') private pivotModel: Model<Pivot>,
5      @InjectModel('sensors') private sensorModel: Model<Sensor>,
6      @InjectModel('soilTypes') private soilTypesModel: Model<SoilType>,
7      @InjectModel('crops') private cropsModel: Model<Crop>,
8    ) {}
9
10   async create(createPivotDto: CreatePivotDto): Promise<Pivot> {
11     const createdPivot = new this.pivotModel(createPivotDto);
12     await createdPivot.save();
13
14     return await this.findBySerialNumber(createPivotDto.serialNumber);
15   }
16
17   async update(updatePivotInput: UpdatePivotDto): Promise<Pivot> {
18     return await this.pivotModel
19       .findOneAndUpdate(
20         {
21           _id: updatePivotInput._id,
22         },
23         {
24           name: updatePivotInput.name,
25           soilType: updatePivotInput.soilType,
26           crop: updatePivotInput.crop,
27         },
28         { new: true },
29       )
30       .populate('soilType', '', this.soilTypesModel)
```

```
31        .populate('crop', '', this.cropsModel);
32    }
33
34    .
35    .
36    .
37 }
```

Listado 5.3: Fragmento de código de la clase PivotsService.

5.3. RF-01: Administración de la plantación, crear pivotes y asignar sensores

La implementación del primer requisito funcional incluye la creación de pivotes de riego y sensores que se asignan a ellos. Esto requiere la interacción entre el cliente web y el servidor a través de la API definida y utilizando la información introducida por el usuario en los formularios correspondientes. En esta sección se describe la implementación de estos formularios y la validación y envío de los datos para cada tarea.

5.3.1. Crear y editar pivotes

Para la creación de nuevos pivotes por parte del administrador de la plantación se añade un formulario utilizando *Reactive Forms* de Angular, esto permite definir los campos de entrada y las funciones validadoras de cada uno del modo que se hace en las primeras líneas de código de la Figura 5.4.

```
1  export class CreatePivotFormComponent implements OnInit
2      form = new FormGroup({
3          name: new FormControl('', [Validators.required, Validators.
       minLength(3), Validators.pattern('[A-Za-z0-9 ]+')]),
4          serialNumber: new FormControl('', [Validators.required,
       Validators.minLength(12), Validators.maxLength(12), Validators.
       pattern('([0-9]|[A-F]|[a-f])*')]),
5          soilType: new FormControl('', [Validators.required]),
6          crop: new FormControl(),
7      });
8
9      .
10     .
11
12   private initializeFormValues() {
13     this.form.get('name')?.setValue(this.data.pivot.name);
14     this.form.get('serialNumber')?.setValue(this.data.pivot.
       serialNumber);
```

```
15      this.form.get('soilType')?.setValue(this.data.pivot.soilType?._id)
        ;
16      this.form.get('crop')?.setValue(this.data.pivot.crop);
17    }
18
19    private async initializeSoilTypesOptions() {
20      this.soilTypeOptions = await this.apiService.getSoilTypes();
21    }
22
23    private async initializeCropOptions() {
24      this.cropOptions = await this.apiService.getCrops();
25      const cropControl = this.form.get('crop');
26
27      if(!cropControl) {
28        return;
29      }
30
31      this.cropFilteredOptions$ = of(this.cropOptions);
32      this.cropFilteredOptions$ = cropControl.valueChanges.pipe(
33        startWith(''),
34        map(input => this.filterCropOptions(input || ''))
35      )
36    }
```

Listado 5.4: Definición del formulario para crear un pivote en el componente CreatePivotFormComponent.

Las opciones seleccionables por el usuario en la elección del tipo de suelo y del cultivo se obtienen mediante dos llamdas HTTP a la API del servidor. En el caso de los cultivos se implementa un campo del formulario donde aparece el listado de todos los tipos disponibles pero se van filtrando a medida que el usuario teclea en ella, con el objetivo de facilitar la selección del valor deseado entre un listado largo de opciones. Para hacerlo se utiliza el atributo *cropFilteredOptions$* de tipo *ObservableCrop[]* , dentro de la función *initializeCropOptions* se utiliza el observable *cropControl.valueChanges* para realizar el filtrado de las opciones cada vez que el usuario modifica el campo '*crop*'.

En el caso de que la entrada de uno de los campos del formulario no cumpla con el formato esperado se inhabilita la opción de enviar y se muestra un mensaje bajo el campo correspondiente indicando el motivo del error. Por ejemplo, en el código de la Figura 5.5 se muestra el código encargado de manejar los posibles errores al introducir el número de serie de un pivote.

```
1     <mat-form-field class="w-full lg:w-1/2">
2         <mat-label>Serial number</mat-label>
3         <input name="serialNumber" matInput formControlName="
    serialNumber" type="text"/>
4
5         @if (form.get('serialNumber')?.hasError('required')) {
6             <mat-error>Serial Number is <strong>required</strong></mat
    -error>
7         }
8
```

```
9           @if (form.get('serialNumber')?.hasError('minlength') || form.
    get('serialNumber')?.hasError('maxlength')) {
10              <mat-error>Serial number must be a <strong> 12 characters
    long.</strong></mat-error>
11          }
12
13          @if(form.get('serialNumber')?.hasError('pattern')) {
14              <mat-error>Serial numbers must contain only hexadecimal
    characters.</mat-error>
15          }
16
17      </mat-form-field>
```

Listado 5.5: Campo serialNumber en el formulario de creación de un pivot.

Para editar un pivote se utiliza el mismo componente pasándole como entrada el estado del pivote e inicializando, en la función *initializeFormValues* el formulario con los datos correspondientes a cada campo.

Cuando el usuario completa la información del formulario y hace clic en *"Submit"*, se realiza una petición de tipo POST (o PATCH, en caso de que esté editando un pivote ya existente) a la ruta *"/pivots"* del servidor y este se encargará de comprobar que no exista otro pivote con ese número de serie y de almacenarlo en base de datos si los datos introducidos son correcto. La respuesta que se recibe en el cliente como resultado de esta operación se mostrará al usuario mediante un mensaje temporal en la parte inferior de la pantalla.

5.3.2. Asignar un sensor a un pivote

De forma similar a como se crea el formulario para crear un nuevo pivote, en el componente *CreateSensorFormComponent* se declara uno con únicamente dos campos, el número de serie del sensor que se quiere añadir y su ángulo dentro de la circunferencia que dibuja el pivote en el que se encuentra instalado, en este caso, es necesario hacer las comprobaciones del formato del número de serie y asegurar que el ángulo introducido está entre 0 y 360 grados.

Finalmente se realiza la petición POST en la ruta *"/pivots/NUM_SERIE_PIVOT/add-sensor"* indicando el número de serie del pivot al que se va a asignar ese sensor. El servidor, de nuevo realizará la comprobación del número de serie y añadirá al listado de sensores de ese pivot el nuevo documento.

5.4. RF-02: Consulta del estado de la plantación en tiempo real.

El flujo de los datos en tiempo real de la plantación empieza en los sensores de humedad instalados en los diferentes pivotes, estos, de forma periódica enviarán los datos leídos al servidor utilizando sockets TCP al servidor. Para ello, cada dispositivo tendrá que haber sido configurado de manera previa para que esos mensajes se dirijan a la dirección correcta y en el puerto definido por el servidor. En este apartado se va a describir como se manejan los datos a partir del momento en el que se reciben en el servidor y como se maneja el estado de cada sensor en el cliente para que todos los componentes interesados se actualicen cuando se recibe nueva información.

5.4.1. Servidor de sockets TCP

Cuando se ejecuta el servidor, este pone a disposición de los sensores un puerto donde enviar los datos del campo. Este puerto se configura en el archivo *.env* del proyecto y debe coincidir con el que los sensores tienen configurado en el firmware.

Se ha definido como requisito, que, cuando un nuevo dispositivo quiera abrir una conexión con el servidor, este debe mandar un mensaje de identificación que incluya su número de serie. De esta manera el servidor podrá relacionar cada socket con el sensor conectado. Se ha extendido la interfaz *Socket* de *NodeJS* para que almacene ese atributo, como puede verse en la figura 5.6, de manera que el servidor podrá actualizarlo cuando reciba ese primer mensaje de identificación. El socket entonces, se almacena en un mapa con su número de serie como clave, para que, cuando el servidor quiera comunicarse con un sensor concreto pueda acceder a su socket de forma directa. Además, los mensajes de datos no necesitarán incluir ese valor pues el servidor ya reconoce a que dispositivo pertenecen.

```
1 export interface SensorSocket extends Socket {
2   serialNumber: string;
3 }
```

Listado 5.6: Interfaz SensorSocket.

Para cumplir con el requisito no funcional RNF-03 (Adaptabilidad a nuevos tipos de sensores), se ha definido la interfaz *DataParser*, que define los métodos utilizados a la hora de interpretar los mensajes entrantes. Para la comunicación con el simulador con el formato de mensajes definido en la Sección 4.4 se implementa esta interfaz en la clase *SimulatorDataParser*. De la misma manera podrían implementarse más protocolos de comunicación y ampliar el tipo de sensores que soporta el servidor. Cuando se recibe un nuevo mensaje, la clase *DataParserFactory* es la encargada de devolver la instancia de *DataParser* necesaria para interpretar ese tipo de datos.

Al recibir un mensaje de de datos desde un sensor, este genera un objeto de tipo *LastState* para cada atributo recibido, este objeto se almacena en base de datos y se envía al Gateway de websockets que comunican con el cliente. Para almacenar ese objeto, se realiza una operación *upsert* en la tabla *LastState* de manera que, si ya existe un documento para ese atributo y número de serie se actualice y si no, se guarde el nuevo objeto. En la tabla *ValueChange* se guardan directamente todos los datos recibidos.

```
private onData(data: Buffer, socket: SensorSocket): void {
    const message = data.toString();

    try {
      const parser = this.dataParserFactory.getParser(message);

      if (parser.isMessageIdentification(message)) {
        const socketId = parser.getSocketIdentification(message);
        socket.serialNumber = socketId;
        this.addSocket(socketId, socket);
        this.sendAck(socket);
        return;
      }

      parser.processMessage(socket.serialNumber, message);
      this.sendAck(socket);
    } catch (err) {
      console.error(err);
    }
  }
}
```

Listado 5.7: Recepción de un nuevo mensaje en el servidor.

5.4.2. Websockets

La comunicación de los datos recibidos de los sensores con el cliente se hace a través de *Websockets*, en la clase *ClientSocketsGateway* del backend se hace uso del decorador de NestJS *@WebSocketGateway* para crear y configurar el servidor de este tipo de socket en el puerto 8080.

El servidor utiliza el concepto de *salas* para mandar los datos de cada sensor a los clientes que se suscriban a la que le corresponde utilizando el número de serie como identificador. El cliente, una vez conoce los sensores que tiene disponibles, mandará un mensaje de tipo *joinRooms* para suscribirse a las salas que le interesan. Finalmente, cada vez que se reciba un mensaje de datos por los sockets TCP se emitirán a la sala correspondiente a ese sensor los objetos *LastState* generados y se recibirá en todos los clientes que se hayan suscrito a ella.

```
@WebSocketGateway(8080, {
  cors: {
    origin: '*',
  },
})
export class ClientSocketsGateway
  implements OnGatewayConnection, OnGatewayDisconnect
{
  @WebSocketServer()
  server: Server;

  @SubscribeMessage('joinRooms')
  handleJoinRooms(
    @MessageBody() data: { rooms: string[] },
    @ConnectedSocket() client: Socket,
  ): WsResponse<unknown> {
    client.join(data.rooms);
    return { event: 'joinRooms', data };
  }

  emitDeviceLastState(lastState: LastState) {
    this.server
      .to(lastState.serialNumber)
      .emit('last-state', JSON.stringify(lastState));
  }
}
```

Listado 5.8: Servidor de Websockets.

5.4.3. Gestión del estado en tiempo real en el cliente

Para manejar el estado en tiempo real de cada sensor en el frontend se utiliza la librería NgRx, esta permite declarar un estado global al que los componentes interesados pueden suscribirse y recibir los nuevos datos cuando estos cambien. En el directorio *src/app/shared/states* se definen las acciones, reductores y selectores necesarios para la gestión del estado de los sensores de humedad, ese estado será esencialmente un listado de los últimos datos recibidos para cada sensor y cada atributo.

Las acciones implementadas se muestra en la Figura 5.9, estás serán emitidas desde los componentes encargados de inicializar el estado o el servicio de sockets del cliente cada vez que recibe un dato nuevo.

```
    export const SensorStateSocketActions = createActionGroup({
    source: 'Websockets',
    events: {
        'New state received': props<{sensorState: SensorState}>()
    }
});

```

```
export const SensorStateApiActions = createActionGroup({
    source: 'Server API',
    events: {
        'Retrieved initial sensor state': props<{states: ReadonlyArray
   <SensorState>}>(),
    }
});
```

Listado 5.9: Acciones de NgRx para la gestión de los datos recibidos desde los sensores.

Los reductores, al detectar estos eventos ejecutan la lógica necesaria para generar un estado nuevo a partir del que se tiene en ese momento. Cuando se recibe *SensorStateApiActions*, lo que se hace es inicializar el estado al que se recibe en el evento, ya que se emite en el componente raíz del panel de control al cargar los datos iniciales de la colección *LastState* desde la API del servidor. Luego, cuando se recibe un dato nuevo por Websockets, el servicio emitirá la acción *SensorStateSocketActions*, en este caso, la función reductora actualizará el estado asegurándose que no existan nunca dos valores para el mismo atributo en el mismo sensor de forma que se almacene únicamente el dato más reciente.

```
    export const initialState: ReadonlyArray<SensorState> = [];

export const sensorStateReducer = createReducer(
    initialState,
    on(SensorStateApiActions.retrievedInitialSensorState, (_state, {
   states}) => {
        return states
    }),
    on(SensorStateSocketActions.newStateReceived, (state, {sensorState
   }) => {
        return state.map(s => {
            if(s.attr === sensorState.attr && sensorState.serialNumber
    == s.serialNumber) {
                return {
                    ...s,
                    value: sensorState.value
                }
            }
            return s;
        })
    })
);
```

Listado 5.10: Reductor para el manejo del estado de los sensores en el cliente.

Cuando un componente, como puede ser *SensorStatusGraphComponent* necesita unos atributos concretos para un sensor puede acceder a ellos utilizando los observables que devuelven las funciones selectoras definidas el fichero *sensor-state.selector.ts* como se hace en la Figura 5.11, estos observables emitirán un nuevo valor siempre que se reciba un dato nuevo para el sensor y los atributos seleccionados por lo que la información se mantendrá actualizada en los componentes suscritos.

```
selectStateByDepth(): void {
    const status$ = getStateObservablesBySnAndAttributes(this.store,
   this.sensor.serialNumber, ['sen1', 'sen2', 'sen3']);
    combineLatest(status$).subscribe((values: number[]) => {
      this.senValues = values as [number, number, number];
    })
  }
```

Listado 5.11: Consulta de atributos en un sensor concreto utilizando el selector de NgRx.

5.5. RF-03: Soporte para la configuración de riegos por sectores dinámicos.

Al hacer clic en el icono de la gota de agua en el panel de un pivot se abre la "Irrigation tool", en ella se puede ver información detallada del tipo de suelo de ese pívot y el estado actual de todos los sensores asociados.

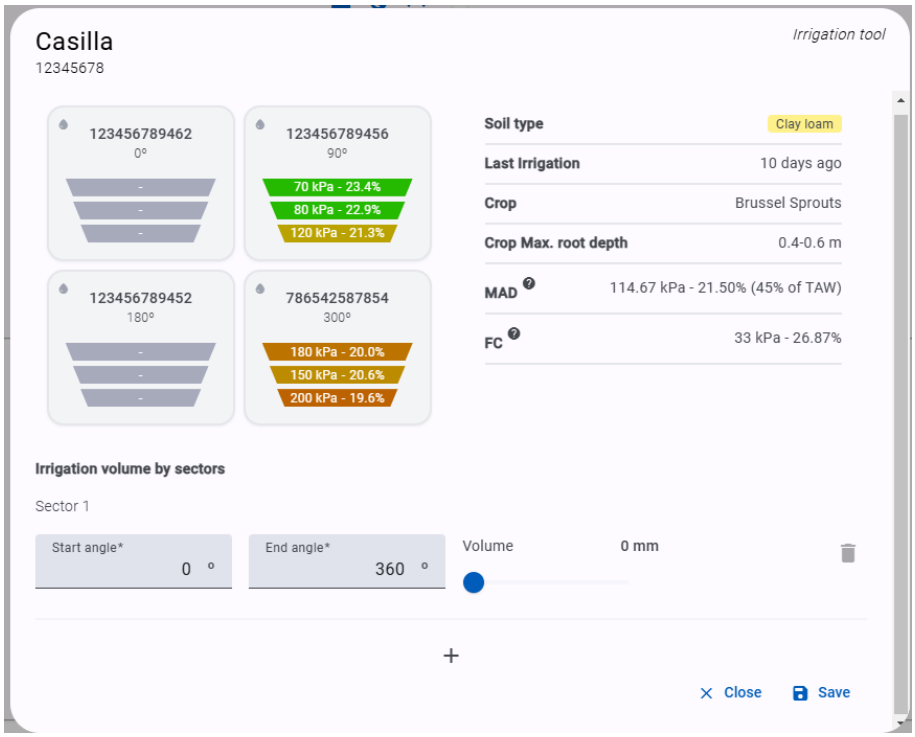

Figura 5.1: Herramienta "Irrigation tool".

Bajo el título "Irrigation volume by sectors" se encuentra el formulario para introducir el riego que se desea aplicar. Se pueden añadir hasta cuatro sectores indicando el ángulo inicial y el final de cada uno así como la cantidad de agua que se desea aplicar en cada uno en mm (litros

por metro cuadrado). Los sectores no pueden solaparse y se mostrará un error al usuario si lo hacen.

Para calcular el valor de la humedad resultante de aplicar cierto volumen de agua en cada sensor se hace uso del atributo θ_s de la Tabla 2.1 como valor máximo de humedad que puede alcanzar ese tipo de suelo, además una vez se ha alcanzado ese valor el exceso de agua se filtrará hacia niveles más profundos por efecto de la gravedad. En cambio, si el agua que se aplica a cierto nivel no es la suficiente como para saturar el suelo, el valor de humedad resultante se calcula como la suma del contenido de agua actual y la fracción de agua respecto a volumen de suelo que representa el riego aplicado, $\theta_{Ii} = \theta_i + V_i \cdot h_i$, donde θ_{Ii} es el CVA resultado en la profundidad i, θ_i es la humedad actual en ese nivel, V_i es el volumen de agua que se aplica en esa profundidad en mm y h es la profundidad de cada nivel. El código para realizar estos cálculos sobre un sensor y una profundidad concreta puede encontrarse en la Figura 5.12.

```
1  private getValueWithIrrigation(senIndex: number) {
2      const sensorIrrigations = this.irrigation?.filter(sector => {
3        return this.sensor.angle >= sector.angleStart && this.sensor.
       angle <= sector.angleEnd
4      });
5
6      // If there aren't any irrigations for this sensor, return the
       current value
7      if(!sensorIrrigations || sensorIrrigations.length === 0) {
8        return this.senValues[senIndex];
9      }
10
11     const sensorIrrigation = sensorIrrigations[0];
12     let irrigationVolume = sensorIrrigation.volume;
13     for(let i = 0; i <= senIndex; i++) {
14       if(irrigationVolume <= 0) {
15         return this.senValues[senIndex];
16       }
17
18       const waterContent = WaterContentService.getWaterContent(this.
       senValues[i], this.soilType);
19       const waterContentToSaturation = this.soilType.
       water_content_saturated - waterContent;
20       const waterToSaturation = waterContentToSaturation * 200;
21
22       if(i === senIndex) {
23         const totalIrrigationWaterContent = irrigationVolume / 200;
24         if(waterContent + totalIrrigationWaterContent < this.soilType.
       water_content_saturated) {
25           return WaterContentService.getPresFromWaterContent(
       totalIrrigationWaterContent + waterContent, this.soilType);
26         } else {
27           return WaterContentService.getPresFromWaterContent(this.
       soilType.water_content_saturated, this.soilType);
28         }
29       } else {
30         // The excess water for the current level filters to the next
       one
31         irrigationVolume -= waterToSaturation;
```

```
32        }
33      }
34
35      return this.senValues[senIndex];
36    }
```

Listado 5.12: Cálculo del contenido de agua al aplicar un riego a cierta profundidad.

Una vez el usuario ha calculado el riego que quiere aplicar al campo puede guardarlo usando en botón "Save" para más tarde evaluar como se ha comportado la humedad del suelo en la realidad utilizando la herramienta de análisis de la evolución de los datos.

5.6. RF-04: Análisis de la evolución de la humedad del campo.

Los gráficos de evolución de la humedad se crean en el componente *DataEvolutionChartComponent* con la librería *amCharts*, utilizando tres series de tipo *LineSeries* para los datos del contenido de agua en el suelo en SMP y una *ColumnSerie* para representar el volumen de los riegos guardados por el usuario.

Los datos del gráfico empiezan a cargase, de forma paginada, en cuanto se hace clic en el botón *"View data evolution"* del panel de un pivote. El servidor mandará los datos de humedad de la colección *ValueChange* de todos los sensores asignados a este en orden de fecha decreciente, de esta manera los primeros datos en llegar son los más recientes y pueden dibujarse en cuanto se reciben, después se van acumulando en el atributo *senData* en orden creciente para evitar fallos visuales en el gráfico.

Durante la carga de los datos el usuario puede ver los datos a medida que se van recibiendo y puede hacer zoom en el gráfico pero no puede navegar entre los diferentes sensores ni cambiar el rango de fechas de los datos representados, esto se controla con el atributo *isLoading*, cuando este se pone a *true* esas opciones se deshabilitan. Una vez se termina el proceso de carga de la información y puede determinarse el periodo de tiempo para el que se tienen datos, ese atributo vuelve a *false*, se habilitan las opciones comentadas y se termina el proceso de inicialización.

```
1  private async getSoilMoistureEvolution() {
2      this.isLoadingData = true;
3      let responseData = [];
4      let offset = 0;
5      const serialNumbers = this.sensors.map(s => s.serialNumber);
6
7      do {
8        const data = await this.apiService.getSoilMoistureEvolution(JSON
     .stringify(serialNumbers), offset, this.DATA_COUNT_LIMIT);
9        responseData = data.map(this.parseResponseItem);
```

```
10
11        if(responseData.length > 0 && !this.fromDate && ! this.toDate) {
12          this.initializeDates(responseData[0]);
13        }
14
15        this.senData.unshift(...responseData.reverse());
16        this.updateData();
17        offset += this.DATA_COUNT_LIMIT;
18      } while(responseData.length > 0);
19
20      if(this.senData.length > 0) {
21        this.minDate = new Date(this.senData[0].createdAt);
22      }
23
24      this.isLoadingData = false;
25    }
```

Listado 5.13: Petición de forma paginada de los datos de humedad del suelo.

Cuando un usuario selecciona otro sensor dentro del mismo pivote, se filtran los datos pasarle al gráfico solo los datos de ese sensor y este se actualiza para representar los nuevos valores. Se hace lo mismo cuando se selecciona un nuevo rango de fechas.

```
1  private updateSoilMoistureData() {
2      const data = this.senData.filter(d => {
3        const itemDate = new Date(d.createdAt);
4        return !this.toDate || !this.fromDate ? false : d.serialNumber
     === this.activeSensor.serialNumber && itemDate <= this.toDate &&
     itemDate >= this.fromDate;
5      });
6
7      this.soilMoistureSeries.forEach(s => s.data.setAll(data));
8    }
```

Listado 5.14: Filtrado de los datos entre las fechas seleccionadas para el sensor activo.

En este gráfico de evolución de la humedad se incluye también un rango, dibujado en color verde, que representa los límites en los que es recomendable mantener la humedad del suelo en ese pivote teniendo en cuenta el cultivo y el tipo del terreno de este. El límite superior es el FC que se toma como 33 kPa y el límite inferior es el MAD, calculado a partir de la fracción de depleción máxima que permite el cultivo y utilizando el modelo de Van Genuchten para convertir el resultado a SMP.

5.7. Simulador

El simulador se ha desarrollado en NodeJS a partir de las necesidades descritas en la Sección 4.4. Además, se ha añadido un script adicional para añadir datos reales a los dispositivos especificados en el archivo *config/devices.ts* que es útil a la hora de desarrollar los gráficos de evolución de la humedad del suelo y sirve también para dejar la aplicación en un estado que puede servir de demostración.

Los datos utilizados para inicializar los sensores se han extraído de la International Soil Moisture Network (ISMN n.d.), específicamente de la estación catalana de El Miracle. Al ejecutar el comando *'npm run seed'* en la raíz del proyecto del simulador se leen estos datos, almacenados en formato CSV en */config/seed-data/*, se convierten de VWC a SMP y se envían a la ruta *lasts-states/seed* del servidor para que este los almacene.

Para simular las comunicaciones con los sensores de humedad instalados en el campo se ha creado la clase *SensorSimulator*, esta contiene un atributo *status* que representa el estado de la humedad del suelo en ese dispositivo. Cuando se llama al método *start* se crea un nuevo socket TCP con el servidor y se envía, una vez confirmada la conexión, el estado inicial del sensor.

```
1   export class SensorSimulator {
2   private socket: Socket;
3
4   private _serialNumber: string;
5
6   private _soilType: SoilTypeName;
7
8   private _status!: [number, number, number];
9
10  private _connected: boolean = false;
11
12  .
13  .
14  .
15
16  constructor(serialNumber: string, soilType: SoilTypeName) {
17      this._serialNumber = serialNumber;
18      this.socket = new Socket();
19      this._soilType = soilType;
20  }
21
22  start() {
23      console.log(`Creating ${this.serialNumber} sensor simulator
    ...`);
24      this.registerListeners();
25      this.connect();
26  }
27
28  private connect() {
29      if(process.env.TCP_SOCKET_PORT && process.env.TCP_SOCKET_HOST)
    {
```

```
30              console.log('Connecting...');
31              this.socket.connect(Number(process.env.TCP_SOCKET_PORT),
    process.env.TCP_SOCKET_HOST);
32          }
33      }
34
35      private registerListeners() {
36          this.socket.on('connect', this.onConnect.bind(this));
37          this.socket.on('data', this.onData.bind(this));
38      }
39
40      private onConnect() {
41          console.log(`Sensor ${this.serialNumber} connected to server
    .`);
42          this._connected = true;
43          this.socket.write(`#SENSOR_CONNECTION:${this.serialNumber}`);
44      }
45
46      private onData(data: Buffer) {
47          console.log('DATA RECEIVED', data.toString());
48      }
49
50      sendStatus() {
51          if(!this.status) {
52              return;
53          }
54
55          const [val1, val2,val3] = this.status;
56          this.socket.write(`#DATA:sen1=${val1},sen2=${val2},sen3=${val3
    }`);
57      }
```

Listado 5.15: Clase SensorSimulator.

Desde el archivo *main.ts*, se crean tantas instancias de *SensorSimulator* como dispositivos se hayan definido en *config/devices.ts*, también se leen los estados iniciales del fichero *sensor-status.txt* y se asignan a sus respectivos simuladores. Todos los datos se mandarán al servidor de forma periódica, pero además, el programa principal se mantiene a la escucha de cambios en ese archivo y volverá a mandarlos cuando el usuario haga alguna modificación en él. Hay ocasiones en las que ese evento puede registrarse dos veces seguidas, por lo que esta funcionalidad se implementa con un pequeño tiempo de *debounce* para evitar así leer y mandar los datos dos veces seguidas, como puede verse en la función *watchStatusFile* de la Figura 5.16.

```
1       private readSensorsStatus() {
2           const statusFileContent = fs.readFileSync(
    SENSORS_STATUS_FILENAME);
3           const sensorsStatus = this.parseStatus(statusFileContent.
    toString());
4           this.setSensorsStatus(sensorsStatus);
5       }
6
7       private sendSensorsStatus() {
8           this.simulators.forEach(sim => {
```

```
9             if(sim.connected) {
10                sim.sendStatus();
11            }
12        });
13    }
14
15    private sendStatusInterval() {
16        setInterval(()=> this.sendSensorsStatus(), this.
      SEND_STATUS_INTERVAL);
17    }
18
19    private watchStatusFile() {
20        fs.watch(SENSORS_STATUS_FILENAME, (eventType) => {
21            if(this.fileChangeDebounceTimeout) {
22                clearTimeout(this.fileChangeDebounceTimeout);
23            }
24
25            this.fileChangeDebounceTimeout = setTimeout(() => {
26                console.log('Status files changed...');
27                this.readSensorsStatus();
28                this.sendSensorsStatus();
29            }, 200);
30        });
31    }
```

Listado 5.16: Envío de los datos de humedad desde el simulador.

5.8. Conclusiones

En este capítulo se ha descrito el proceso de implementación de la aplicación, detallando cómo se desarrollaron las entidades principales, las operaciones CRUD y las rutas HTTP para la gestión de los datos. Utilizando NestJS para el backend y Angular en el frontend, se han desarrollado las tareas que se corresponden con los casos de uso de la Sección 3.3, asegurando que se cumple con el flujo principal de cada uno y controlando sus posibles flujos alternativos.

Además, se ha implementado un simulador en NodeJS que reproduce el comportamiento de los sensores reales, permitiendo validar la solución sin la necesidad de contar con los dispositivos físicos durante la fase de desarrollo.

Capítulo 6

Pruebas

6.1. Introducción

En este capítulo se va a hacer un repaso por las funcionalidades implementadas, teniendo en cuenta como se definieron en los casos de uso del Capítulo 3 y comprobando que cumplen con los requisitos funcionales descritos en este.

6.2. Añadir pivote de riego

El usuario puede abrir el formulario para crear un nuevo pivote haciendo clic en el botón "Add pivot"de la pantalla principal, en este el usuario puede introducir los datos del nuevo equipo. Cuando el nombre de este nuevo dispositivo contiene símbolos o el número de serie no cumple con el formato definido se muestran los errores correspondientes (Figuras 6.1 y 6.2, respectivamente) y se deshabilita el botón de confirmación para evitar el envío de datos erróneos.

Una vez completado el resto de campos el usuario puede enviarlos para añadir el pivote al sistema (Figura 6.3). Al hacer clic en *'Submit'*, y en caso de que el número de serie no exista ya en la base de datos y los datos se almacenen correctamente, se mostrará un mensaje de éxito (Figura 6.4), si no, se mostrará otro indicando el motivo del error (Figura 6.5).

Cuando el flujo de esta funcionalidad termina con éxito, puede comprobarse que el pivote se ha añadido al panel de control y persiste al recargar la página.

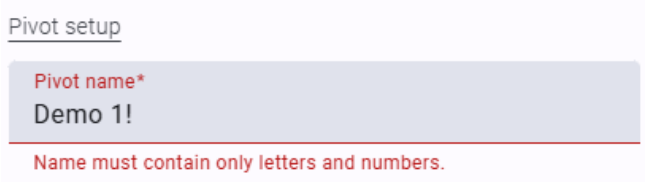

Figura 6.1: Error al introducir un nombre de pivote incorrecto.

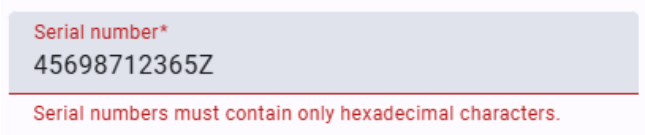

Figura 6.2: Error al introducir un número de serie incorrecto.

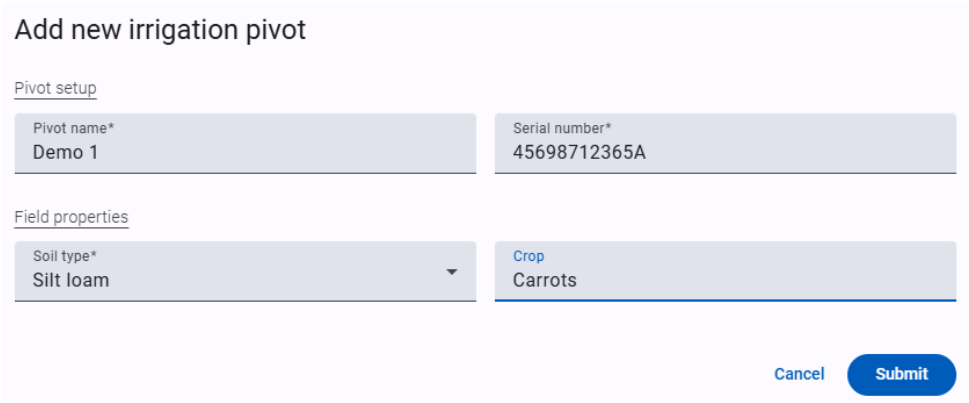

Figura 6.3: Formulario para crear el pivote con datos correctos.

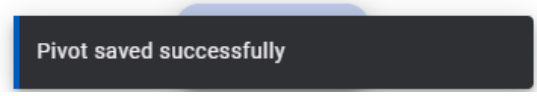

Figura 6.4: Mensaje al crear con éxito un nuevo pivote.

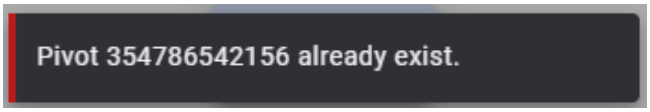

Figura 6.5: Error al crear un nuevo pivote.

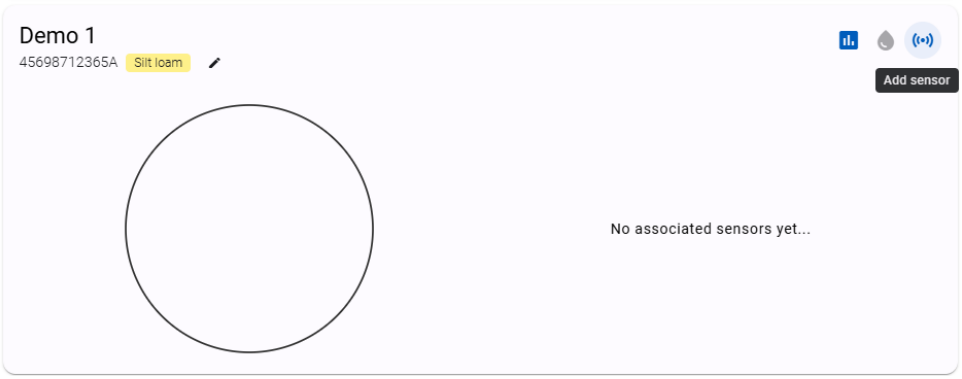

Figura 6.6: Panel de un pivote y botón para añadirle un sensor.

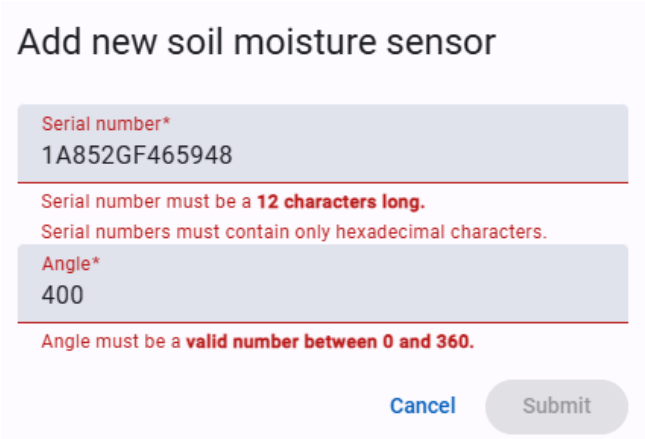

Figura 6.7: Errores al introducir datos inválidos de un nuevo sensor.

6.3. Asignar sensor a pivote de riego

Para asignar un nuevo sensor a un pivote existente, el usuario hace clic en el botón de la esquina superior derecha dentro del panel de este (Figura 6.6), esto abre el formulario correspondiente y permite introducir el número de serie y el ángulo en el que se encuentra este dispositivo dentro del pivote.

De nuevo, al introducir datos que no cumplen con el formato exigido se indica debajo de cada campo el error pertinente y se deshabilita el envío del formulario (Figura 6.7). Cuando el usuario corrija esta información podrá enviarla al servidor. El resultado de la operación se mostrará en pantalla como se muestra en las Figuras 6.8 y 6.9.

Al terminar este proceso el sensor se añade a la base de datos e inmediatamente se actualiza la interfaz para incluir el sensor dentro del pivote correcto (Figura 6.10).

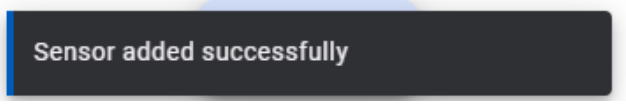

Figura 6.8: Mensaje de éxito al asignar un nuevo sensor.

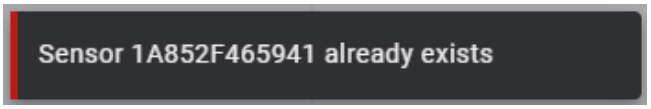

Figura 6.9: Mensaje de error al asignar un sensor que ya existe.

Figura 6.10: Resultado de asignar un sensor a un pivote.

6.4. Recibir datos desde un sensor y actualizar su estado en el panel de control

Para validar este caso de uso se va a utilizar el simulador para mandar los datos al servidor, es necesario entonces configurarlo para comunicar con el sensor que se ha añadido en la Sección 6.3. En el archivo de configuración *simulator/config/devices.ts* se añade un nuevo *SensorSimulator* con el número de serie del sensor y se añade una nueva línea a *simulator/sensor-states.txt* con la cadena *"1A852F465941=50,100,150"* para indicar que el contenido de agua en este sensor será de 50, 100 y 150 kPa a 20, 40 y 60 cm de profundidad, respectivamente.

Al poner en marcha el simulador enseguida puede verse por consola la confirmación de que se ha conseguido conectar con el servidor (Figura 6.11). Pasado un minuto, que es el intervalo de tiempo en el que el simulador manda los datos, en la terminal del servidor puede verificarse la recepción del mensaje con el estado del dispositivo simulado (Figura 6.12)

También puede comprobarse que estos datos se han almacenado en base de datos mirando directamente en las colecciones *LastState* y *ValueChange* de MongoDB (Figura 6.13).

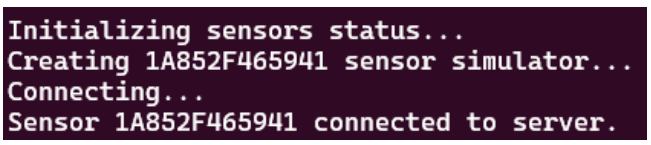

Figura 6.11: Salida por consola al conectar el simulador al servidor.

Figura 6.12: Recepción de un mensaje por socket TCP en el servidor.

```
_id: ObjectId('66ecb54588b29f186cc66ae0')
attr : "sen3"
serialNumber : "1A852F465941"
__v : 0
createdAt : 2024-09-19T23:35:33.051+00:00
updatedAt : 2024-09-19T23:38:14.223+00:00
value : "150"
```

```
_id: ObjectId('66ecb54588b29f186cc66adf')
attr : "sen1"
serialNumber : "1A852F465941"
__v : 0
createdAt : 2024-09-19T23:35:33.050+00:00
updatedAt : 2024-09-19T23:38:14.223+00:00
value : "50"
```


Figura 6.13: Datos de humedad almacenados en la colección *LastState*.

Finalmente, en el navegador web pueden verse los mensajes entrantes a través de Websockets en la pestaña de "Red"de las herramientas de desarrolladores, en esta se muestran las tres tramas correspondientes a cada uno de los niveles de profundidad del sensor (Figura 6.14). Además, el gráfico del sensor en el panel del pivote se actualizará para mostrar los nuevos valores (Figura 6.15).


```
42["last-state","{\"serialNumber\":\"1A852F465941\",\"attr\":\"sen1\",\"value\":\"50\"}"]
42["last-state","{\"serialNumber\":\"1A852F465941\",\"attr\":\"sen2\",\"value\":\"100\"}"]
42["last-state","{\"serialNumber\":\"1A852F465941\",\"attr\":\"sen3\",\"value\":\"150\"}"]
```


Figura 6.14: Mensajes entrantes en el navegador a través de Websockets.

Figura 6.15: Panel del pivote actualizado con los nuevos datos.

6.5. Simular riegos y guardar la configuración utilizada

Puede abrirse la herramienta *"Irrigation tool"* haciendo clic en el icono de la gota de agua en el panel del pivote. Dentro de esta se muestra información detallada del pivote que puede resultar útil al usuario a la hora de considerar la cantidad de agua que utilizará en el próximo evento de irrigación: el tipo de suelo, el tipo de cultivo y la profundidad máxima de sus raíces, la fecha

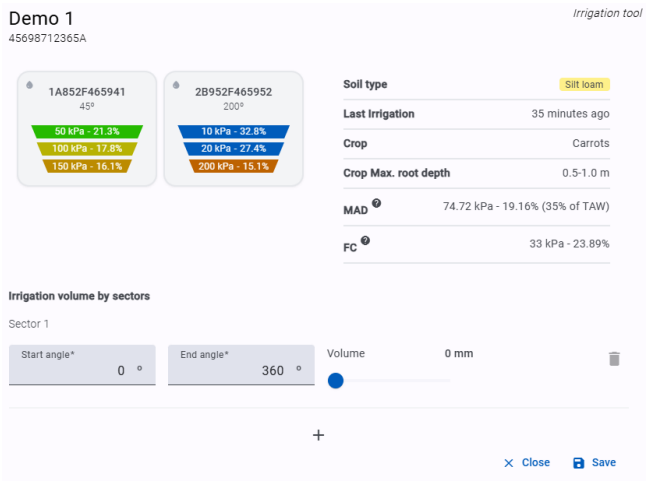

Figura 6.16: Herramienta “Irrigation tool”.

del último riego guardado y los límites FC y MAD que, como se comenta en la Sección 2.5, se consideran el límite superior e inferior para el contenido de agua en el tipo de terreno del pivote y el tipo de planta en él.

En este menú pueden verse también todos los sensores del pivote y su estado actual. Debajo de ellos, se encuentra el formulario para añadir sectores de riego, puede comprobarse que, si se varía la cantidad de agua que se quiere aplicar para la circunferencia completa del pivote, se actualiza el contenido de agua de los sensores para reflejar el resultado del riego con el volumen introducido.

Pueden añadirse hasta cuatro sectores diferentes siempre y cuando estos no se solapen y tengan valores válidos, si sucede uno de estos casos se deshabilita el botón para enviar esa configuración y se muestran los errores de la Figura 6.17 o la Figura 6.18.

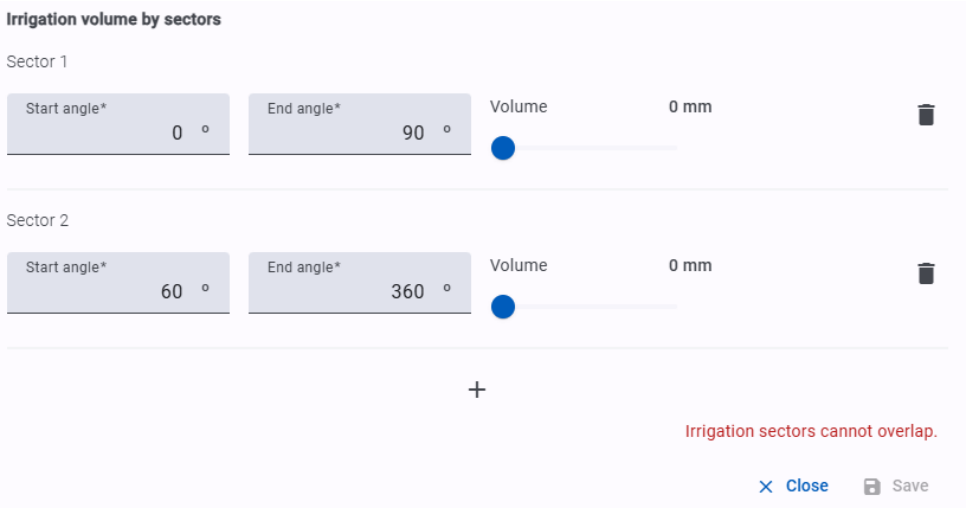

Figura 6.17: Error al intentar crear dos sectores de riego que se solapan.

A la hora de realizar la simulación del resultado del riego en cada sensor se debe tener en cuenta únicamente el sector donde se encuentran, por ejemplo, en la Figura 6.19 se muestran dos sectores de riego, cuyo estado inicial se muestra en la Figura 6.16, con el resultado de aplicar

129 mm de agua entre los ángulos 180 y 360, pero nada en el resto de la circunferencia, como consecuencia, el primer sensor no es afectado por el riego, pero se consigue un suelo totalmente saturado en el segundo sensor.

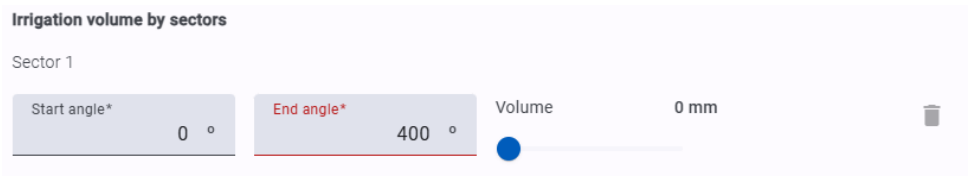

Figura 6.18: Error por ángulo inválido en un sector de riego.

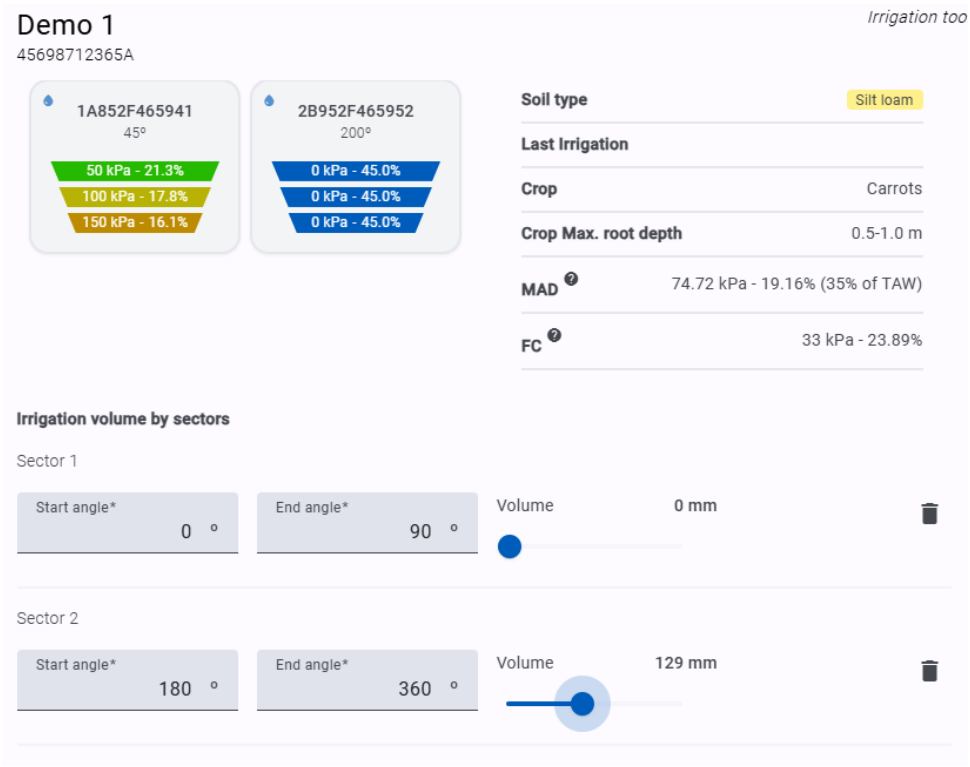

Figura 6.19: Simulación de riego con diferentes sectores.

Una vez el usuario está satisfecho con la configuración del riego hace clic en el botón *Save* para enviarlo al servidor y se muestra un mensaje de éxito una vez se ha guardado el resultado en base de datos. Se comprueba que este se ha almacenado de forma correcta en la colección *Irrigations*.

6.6. Consultar la evolución de la humedad del suelo

En la evaluación de este caso de uso se utiliza el script *seed.ts* del simulador para tener datos que mostrar en los gráficos y demostrar así que se cargan correctamente. De esta manera, cuando se hace clic en el icono del gráfico en el pivote que se creó en la Sección 6.2, se abre la herramienta para visualizar la evolución de la humedad y empieza la carga de datos.

En el gráfico puede verse el riego que se ha añadido al pivote en la Sección 6.5 , sin embargo, como los datos utilizados en el *seeding* son hasta 2023 no se ven datos de humedad hasta que se modifica el rango de fechas utilizando el selector de la parte superior se obtiene el resultado de la Figura 6.20.

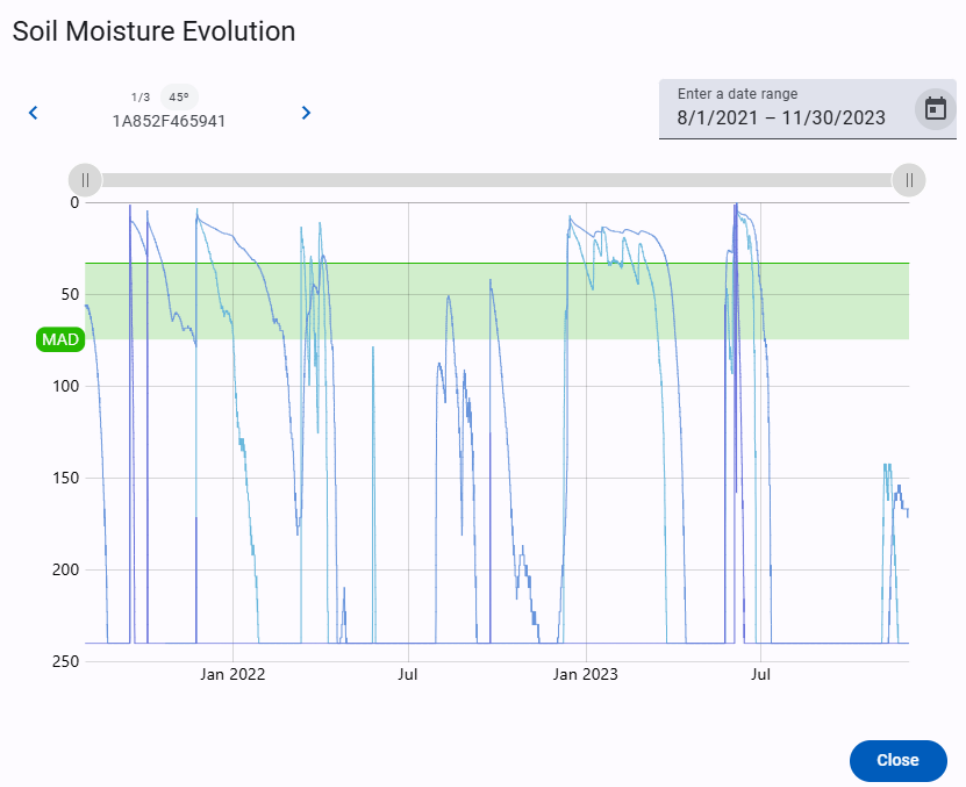

Figura 6.20: Gráfico de evolución de la humedad con datos de la estación El Miracle, de ISMN.

Cuando se cambia de sensor utilizando las flechas también se actualiza el gráfico con los datos del nuevo dispositivo. En el caso de que este no haya comunicado todavía con el sensor se muestra el mensaje *"No data"* (Figura 6.22).

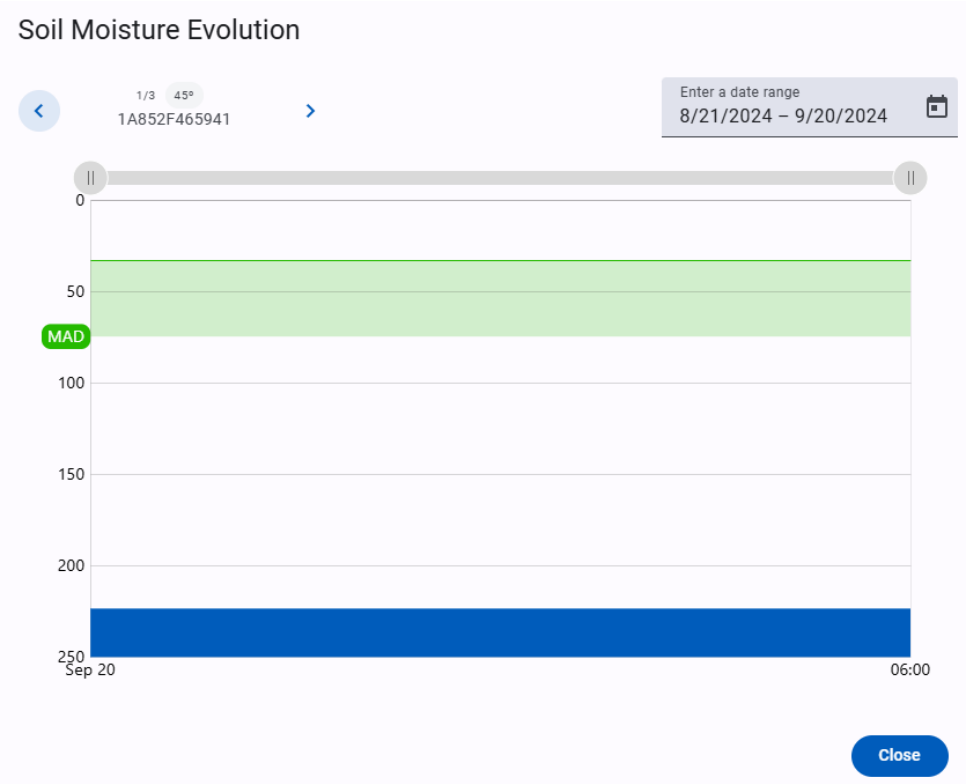

Figura 6.21: Gráfico de evolución de la humedad con un riego.

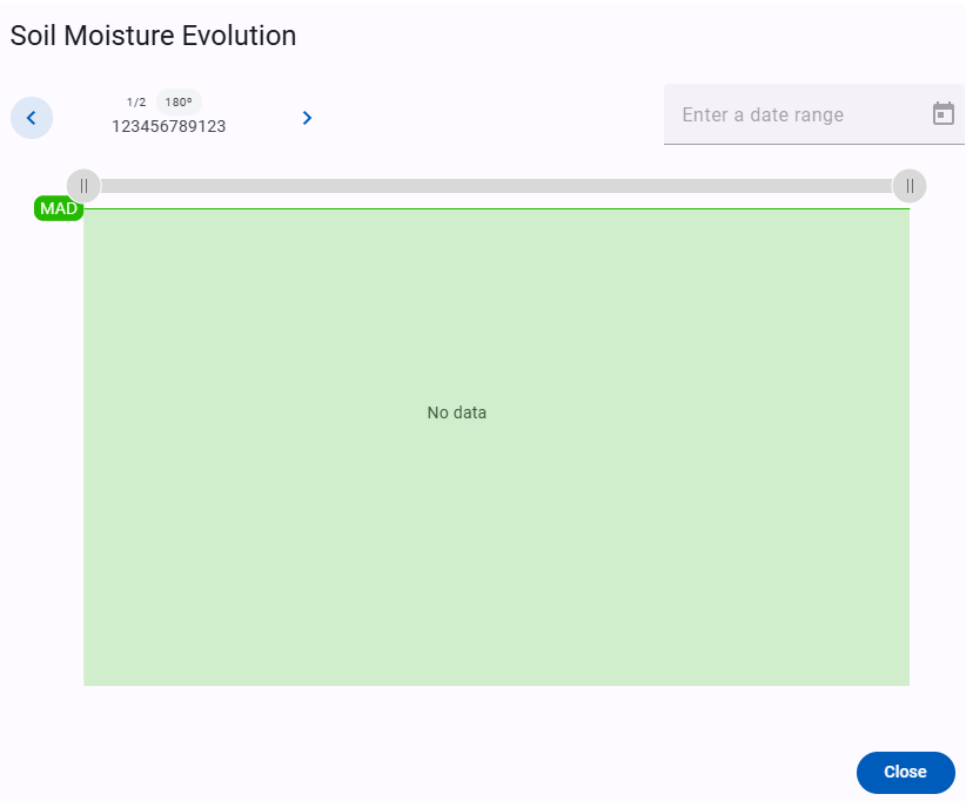

Figura 6.22: Gráfico de evolución de la humedad sin datos.

6.7. Conclusiones

En este capítulo se ha validado que las funcionalidades implementadas cumplen con los requisitos definidos en los casos de uso siendo correctos todos los pasos del flujo principal y las post-condiciones de cada uno. Además, el sistema ha demostrado ser capaz de manejar correctamente los errores y casos secundarios según se describe en los flujos alternativos de cada tarea.

Capítulo 7

Conclusiones y trabajos futuros

7.1. Conclusiones

A partir de la idea inicial de Proxima Systems y los objetivos planteados, se ha desarrollado un sistema de apoyo a la decisión que sirve al gestor de una plantación para facilitar el uso eficiente del agua, lo que puede ser crucial para su rendimiento. Este sistema consiste en un panel de control que se mantiene actualizado con los datos que se reciben de los sensores situados en el campo, que permite la simulación de riegos mediante pivotes y el análisis de la evolución de los datos generados. Las metas del proyecto se han conseguido mediante el estudio en profundidad del problema y su marco teórico, definiendo los requisitos de la solución y evaluando que la herramienta implementada cumple con esa definición inicial.

Sin duda la parte más compleja ha sido la de entender el problema y poder diseñar una solución válida dentro de un campo especializado como es la agricultura. Una vez superada esa fase inicial de aprendizaje (aunque realmente no se ha dejado de aprender en todo el curso del proyecto) se definió de forma clara y sin ambigüedades cuales serían los requisitos funcionales y no funcionales de la herramienta final y como se cumplirían mediante los casos de uso de la Sección 3.3. Con esa descripción, se identificaron las entidades principales de la solución, sus atributos y las relaciones entre ellas. Como parte del análisis, también se definieron los diferentes componentes del sistema y los métodos de comunicación entre ellos de manera que permitieran registar los eventos en tiempo real y almacenar toda la información generada en la aplicación. En esta fase se decidió utilizar los sockets TCP para la comunicación entre los sensores y el servidor y el protocolo de Websockets entre este y los clientes web.

A continuación se lleva a cabo la implementación de forma incremental, empezando por la creación del esquema de base de datos, los servicios y controladores del *backend* sobre lo que se fundamentan todas las demás tareas. Después, se implementan las funcionalidades correspondientes a los casos de uso definidos en la fase anterior y, finalmente, se evaluó el resultado de esa implementación para verificar que se cumplían los flujos principales de todos ellos y se controlaban los posibles flujos alternativos.

7.2. Trabajos futuros

Aunque como resultado de este trabajo se haya obtenido una aplicación que cumple con los objetivos planteados al inicio, vale la pena reflexionar sobre la solución implementada y las mejoras que podrían llevarse a cabo en el futuro.

En cuanto al algoritmo de la simulación de los riegos, como se ha visto en la sección 5.5, se ha decidido utilizar un método que consiste en calcular cuanta agua necesita cierto tipo de suelo para saturarse, sabiendo que el restante percolará hacia niveles más profundos, de esta manera, partiendo del estado inicial de la humedad del suelo medido por los sensores de un pivote y dado un volumen de agua que se desea aplicar, se puede calcular el contenido de agua resultante a diferentes profundidades. Aunque se ha validado junto a Proxima Systems que es un método válido, la exactitud de este método puede verse afectado por varios factores que no se tienen en cuenta para mantener manejable la complejidad de la herramienta. El factor más importante es la evapotranspiración, es decir, la cantidad de agua que se extraerá del suelo por evaporación por causas meteorológicas, principalmente la radiación solar que incide en el suelo y la temperatura ambiental, y la cantidad de agua que transpiran las plantas, almacenándola en sus tejidos y haciendo que se evapore a la atmósfera. En (Allan et al. 1998) se proveen diversos métodos para calcular la evapotranspiración, esta podría incorporarse a la aplicación si se integrara una estación meteorológica en el sistema o utilizando un valor de evapotranspiración de referencia que suministran servicios como el Sistema de Información Agroclimática para el Regadío (SIAR). Una vez que se tuviera este factor de evaporación por causas atmosféricas habría que multiplicarlo por un factor que dependería del tipo de cultivo y de su estado fenológico, por lo que se tendría que tener una tabla con los factores de evapotranspiración para las plantas que quisieran soportarse en la aplicación en diferentes estados de crecimiento, de nuevo, en Allen et al. se proveen algunos de estos valores. Otro de los factores que afecta al cálculo del agua necesaria para los riegos es el agua que se pierde por desprendimiento superficial, causado cuando el riego se aplica con un flujo que es superior a la velocidad a la que el suelo puede absorberlo, esto hará que se formen charcos y ese agua acumulada se evapore más rápido o, en caso de que haya pendiente en el terreno, esta se deslice por ella. En la aplicación desarrollada se da por hecho que los pivotes están configurados de manera que se evite este efecto, pero puede que no siempre sea así, sería útil añadir un cálculo del flujo máximo al que debe funcionar el equipo según el tipo de suelo en el que se instala.

Por otro lado, viendo la complejidad de los cálculos comentados y, aprovechando la cantidad de datos que se coleccionan en una aplicación de este tipo se considera que una solución a futuro ideal es la de incorporar un modelo de aprendizaje automático. Este modelo podría entrenarse para predecir el volumen de agua óptimo para dejar el suelo por debajo de FC a la profundidad máxima de las raíces del cultivo. Este algoritmo podría ser muy preciso si además se incorporara la información necesaria para inferir la evapotranspiración, datos meteorológicos y el estado de crecimiento del cultivo. Aunque esta opción se consideró para este proyecto y se llegó a realizar parte del desarrollo con datos del ISMN se consideró demasiado complejo para el alcance de este trabajo, en cambio, en Proxima Systems está planeado realizar una implementación similar en el largo plazo.

Respecto al diseño de la aplicación, el planteamiento es adecuado en este caso porque se quería

centrar la atención en la comunicación de los sensores con el sistema, la monitorización en tiempo real del estado del campo y en la herramienta de soporte a los riegos, sin embargo, para que este proyecto funcionara en un entorno real sería necesario añadir autenticación de usuarios de manera que cada usuario accediera a un panel de control particular con acceso únicamente a sus dispositivos. También sería conveniente añadir diferentes roles para administrar personas y equipos. Por último, los usuarios deberían ser capaces de eliminar y sustituir dispositivos en caso de que alguno fallara.

Bibliografía

AGRIVI (2024), 'Center pivot irrigation: From dust bowl to modern era methods'.
URL: *https://www.agrivi.com/blog/center-pivot-system-an-efficient-and-economical-solution-for-irrigation/*

Allan, R., Pereira, L. & Smith, M. (1998), *Crop evapotranspiration-Guidelines for computing crop water requirements-FAO Irrigation and drainage paper 56*, Vol. 56, FAO.

Carsel, R. F. & Parrish, R. S. (1988), 'Developing joint probability distributions of soil water retention characteristics', *Water Resources Research* **24**, 755–769.
URL: *https://api.semanticscholar.org/CorpusID:129366485*

Cerf, V. G. & Kahn, R. E. (1974), 'A protocol for packet network intercommunication', *IEEE Transactions on Communications* **22**(5), 637–648.

Datta, S., Taghvaeian, S. & Stivers, J. (2017), 'Understanding soil water content and thresholds for irrigation management', *Oklahoma Cooperative Extension Service* .

Fette, I. & Melnikov, A. (2011), The WebSocket Protocol, Request for Comments RFC 6455, Internet Engineering Task Force (IETF).
URL: *https://www.rfc-editor.org/info/rfc6455*

G. Evans, R. (2010), *Center Pivot Irrigation Design Manual*, 2 edn.

ISMN (n.d.), 'International soil moisture network', https://ismn.earth.

Muñoz Vicente, E. (2019), 'Hasta un 30 % de reducción adicional en el coste energético del riego gracias al ajuste de la presión de bombeo en función del número y las presiones en los pivotes regando en cada momento', https://www.proximasystems.net/agricultura/hasta-un-30-de-reduccion-adicional-en-el-coste-energetico-del-riego/.

Van Genuchten, M. (1980), 'A closed-form equation for predicting the hydraulic conductivity of unsaturated soils1', *Soil Science Society of America Journal* **44**.

W. Eddy, E. (2022), Transmission Control Protocol (TCP) Specification, Standards Track RFC 9293, Internet Engineering Task Force (IETF). Obsoletes: RFC 793, RFC 879, RFC 2873, RFC 6093, RFC 6429, RFC 6528, RFC 6691, RFC 1011, RFC 1122, RFC 5961.
URL: *https://www.rfc-editor.org/info/rfc9293*

Listado de Abreviaturas

SMP Potencial Matricial del Suelo

VWC Contenido Volumétrico de Agua

TCP Transmission Control Protocol

API Interfaz de Programación de Aplicaciones

RF Requisito Funcional

ISMN International Soil Moisture Network

CSV Valores Separados por Comas

FC Capacidad de Campo

MAD Gestión de la Depleción Máxima

PWP Punto de Marchitamiento Permanente

TAW Total de Agua Disponible

FAO Food and Agriculture Organization

CRUD Create, Read, Update, Delete

CPO Cloud Pressure Optimization

Anexo. Manual de usuario

A.1 Instalación y arranque

Antes de proceder con la instalación del sistema, asegúrese de que los siguientes componentes están instalados y correctamente configurados en su equipo:

- Docker
- Docker Compose

Luego, abra una terminal, sitúese en el directorio raíz del proyecto y ejecute:

```
docker compose up --build -d
```

Esto instalará las dependencias y lanzará los diferentes componentes del proyecto: backend, frontend y base de datos. Cuando ese proceso termine, la aplicación web será accesible desde un navegador en la ruta http://localhost:8080/dashboard.

Para detener el sistema, ejecute:

```
docker compose down
```

A.2 Añadir un nuevo pivote de riego

1. En el panel de control principal, haga clic en el botón "Añadir Pivote" (Add Pivot).
2. Se abrirá un formulario donde deberá completar los siguientes campos:
   - **Nombre del pivote**: Asegúrese de utilizar un nombre válido (solo letras, números y espacios).
   - **Número de serie**: Introduzca el número de serie de 16 caracteres en formato hexadecimal.
   - **Tipo de suelo**: Seleccione el tipo de suelo del menú desplegable.
   - **Tipo de cultivo**: Seleccione el cultivo que se plantará en el pivote.

3. Revise que toda la información sea correcta y haga clic en “Guardar”.

4. Si los datos son válidos, el pivote se añadirá al sistema y aparecerá en el panel de control.

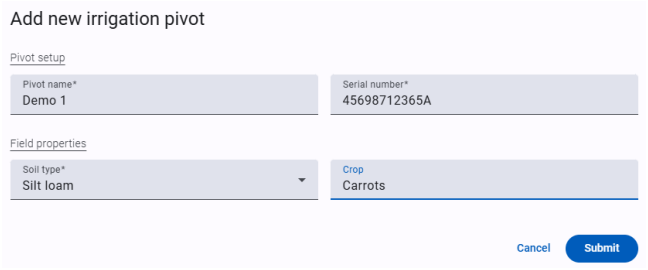

Figura A.1: Formulario de creación de un pivote de riego.

A.3 Asignar sensores a un pivote de riego

1. Seleccione el pivote de riego al que desea asignar sensores desde la lista en el panel de control.

2. Haga clic en el botón “Añadir sensor” (Add sensor) en el pivote seleccionado.

3. Introduzca la siguiente información:

   - **Número de serie del sensor**: Debe tener 16 caracteres hexadecimales.
   - **Ángulo del sensor**: Defina el ángulo en el que se colocará el sensor dentro del pivote, tomando como referencia el norte (0^{o}).

4. Haga clic en “Guardar”. Si la información es correcta, el sensor se asociará al pivote y el sistema actualizará el panel de control.

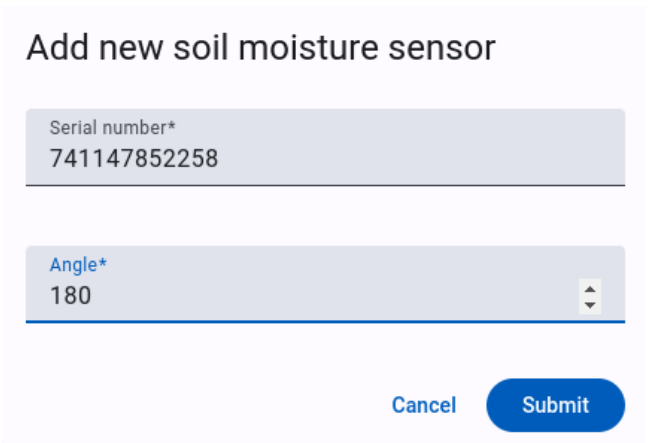

Figura A.2: Formulario para la asignación de un nuevo sensor a un pivote.

A.4 Monitorización en tiempo Real del estado de la plantación

1. En el panel de control, podrá ver el estado de la humedad del suelo de cada pivote en tiempo real.
2. Los sensores envían datos periódicamente sobre los niveles de humedad a tres profundidades diferentes.
3. El sistema muestra un código de colores que indica si el suelo está en niveles óptimos de humedad para el tipo de cultivo seleccionado:
   - **Verde**: Nivel de humedad adecuado.
   - **Amarillo-Rojo**: Riesgo de marchitación del cultivo por sequía.
   - **Azul**: Exceso de agua.

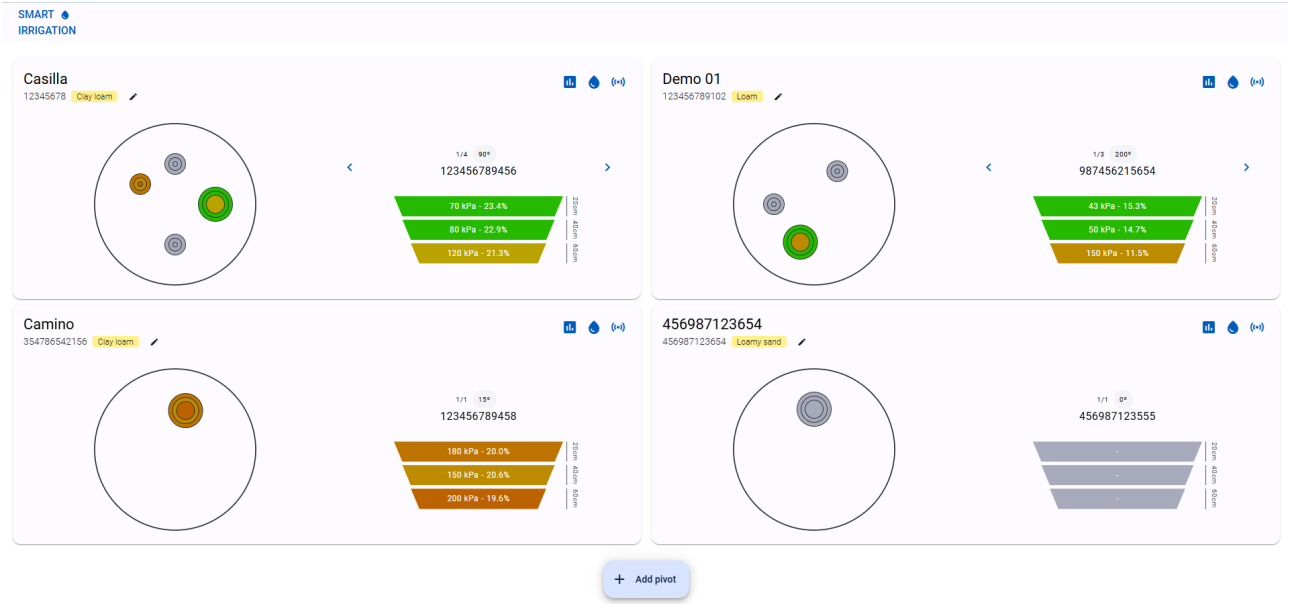

Figura A.3: Panel de control con diversos pivotes y sensores.

A.5 Simulación de riegos y configuración de sectores

1. En el panel del pivote, haga clic en el botón “Herramienta de Riegos” (Irrigation Tool).
2. Defina los sectores de riego introduciendo:
   - **Ángulo inicial y final**: Especifique el ángulo del sector que desea regar.
   - **Volumen de agua**: Introduzca la cantidad de agua en mm para ese sector.

3. El sistema simulará el riego y le mostrará el resultado esperado en los sensores correspondientes, permitiéndole ajustar el volumen si es necesario.

4. Cuando esté satisfecho con la configuración, haga clic en “Guardar” para almacenar los parámetros del riego.

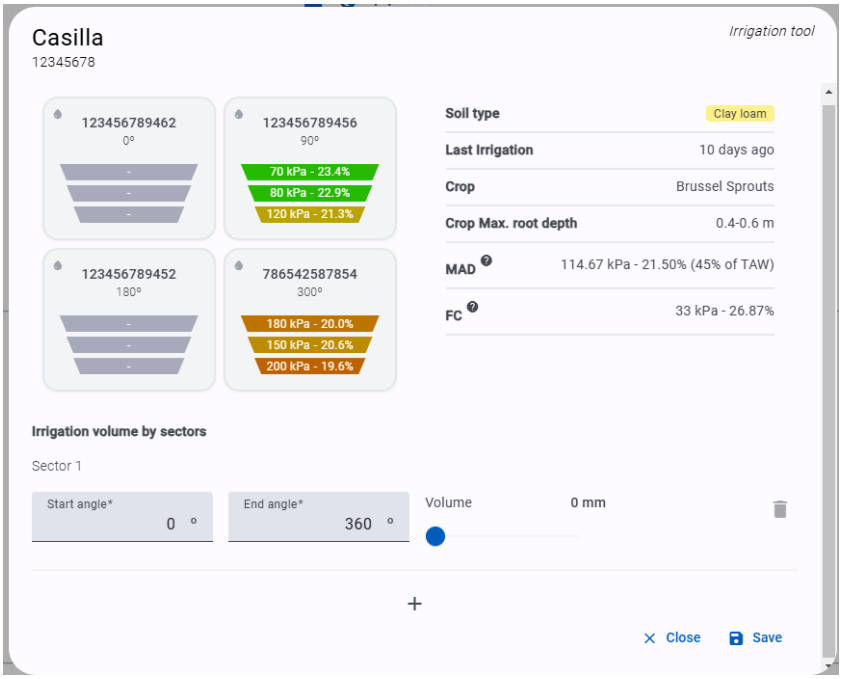

Figura A.4: Herramienta de configuración de riegos

A.6 Consultar la evolución de la humedad del suelo

1. Haga clic en el botón “Ver evolución de los datos” (View data evolution) en el panel del pivote que desea analizar.

2. El sistema le mostrará un gráfico con la evolución de la humedad en las tres profundidades monitorizadas, así como los volúmenes de riego aplicados.

3. Puede ajustar el rango de fechas para observar cómo ha cambiado la humedad a lo largo del tiempo y navegar entre los diferentes sensores del mismo pivote.

Esta herramienta le permite analizar el comportamiento de los riegos aplicados en el terreno, utilizando los rangos de humedad del suelo recomendados para hacer un uso eficiente del agua.

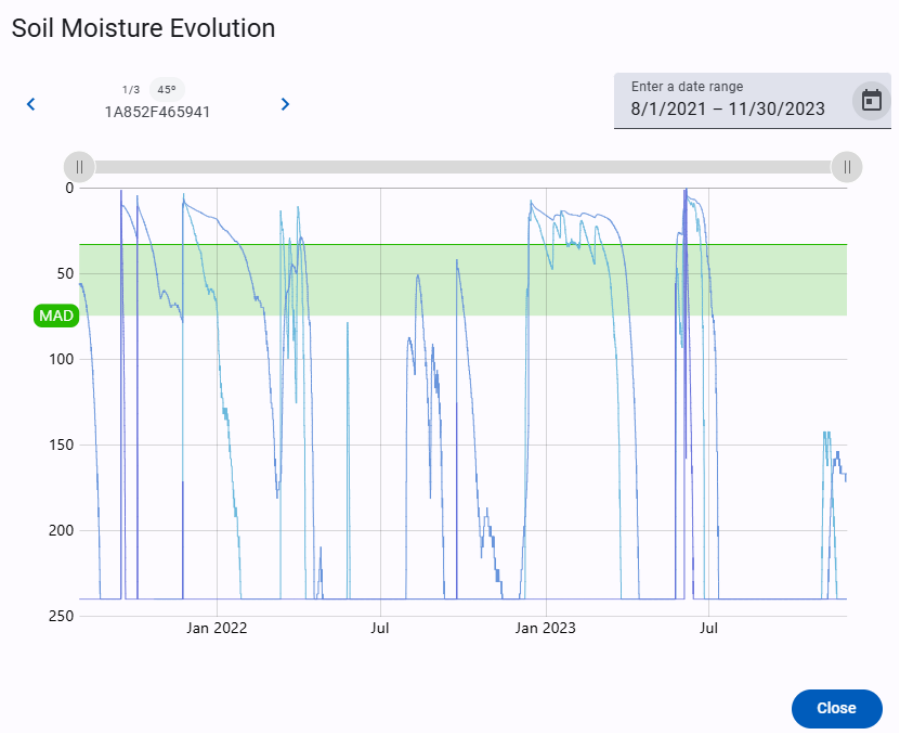

Figura A.5: Gráfico de la evolución de la humedad del suelo en un sensor.

A.7 Simulador de sensores (para desarrolladores)

El simulador está disponible para realizar pruebas y demostraciones del sistema sin necesidad de usar los sensores físicos. Para usar el simulador, siga los pasos a continuación.

Configuración del simulador

El simulador utiliza una configuración basada en ficheros para definir los sensores que desea simular y sus respectivos estados. Para configurar el simulador:

1. Acceda a los archivos de configuración del simulador, ubicados en el directorio `config`.
2. Modifique el archivo `devices.ts`, donde debe definir los números de serie de los sensores que desea simular. A continuación se muestra un ejemplo de cómo añadir un nuevo sensor:

```
export const devices: SensorSimulator[] = [
    new SensorSimulator('786542587854', SoilTypeName.SiltLoam),
];
```

   En este archivo, debe especificar el número de serie del sensor y el tipo de suelo que se va a simular.

3. Después, edite el archivo `sensor-states.txt` para definir los valores de humedad del suelo para cada sensor a diferentes profundidades. Cada línea de este archivo representa el estado de un sensor, donde los valores de humedad están separados por comas. El formato es el siguiente:

   ```
   1A852F465941=50,100,150
   2B123E785643=45,85,120
   ```

   En este ejemplo, el sensor con número de serie `1A852F465941` tiene un valor de 50 kPa a 20 cm de profundidad, 100 kPa a 40 cm, y 150 kPa a 60 cm.

Ejecución del simulador

Para ejecutar el simulador y empezar a enviar datos al servidor, siga estos pasos:

1. Abra una terminal en el directorio raíz del simulador.
2. Asegúrese de que tiene instaladas todas las dependencias utilizando el comando:

   ```
   npm install
   ```

3. Una vez terminado este proceso, puede lanzar el simulador ejecutando el siguiente comando:

   ```
   npm run start
   ```

   Esto iniciará el simulador y se conectará al servidor. Los sensores configurados comenzarán a enviar sus datos periódicamente, tal como lo harían los dispositivos reales en el campo.

4. Si desea añadir datos históricos simulados para pruebas o demostraciones, puede usar el script de inicialización de datos. Es útil si necesita mostrar un gráfico con datos pasados:

   ```
   npm run seed
   ```

Modificación dinámica de los estados del sensor

El simulador también puede detectar cambios en el archivo `sensor-states.txt`. Si desea actualizar los valores de humedad en tiempo real:

1. Abra el archivo `sensor-states.txt`.
2. Modifique los valores de humedad del sensor que desee.
3. Guarde el archivo. El simulador detectará automáticamente los cambios y enviará los nuevos valores al servidor.

Detener el simulador

Para detener el simulador, simplemente presione `Ctrl + C` en la terminal donde se está ejecutando el simulador. Esto cerrará todas las conexiones activas con el servidor y detendrá el envío de datos.