UNIVERSIDAD DE CÓRDOBA

APLICACIÓN DE LAS TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIÓN A LA GESTIÓN SOSTENIBLE DEL RIEGO

TRABAJO FIN DE MÁSTER

Curso 2019-2020

Francisco Puig Pérez-Barquero

Tutores:

Juan Antonio Rodríguez Díaz
Rafael González Perea

Resumen:

El presente trabajo pretende mostrar la metodología para el desarrollo de una aplicación web que calcula un riego de precisión. Para ello, el trabajo comenzó con el diseño de la programación de riego de la finca de estudio. Dicha finca consta de 44 hectáreas de almendros en intensivo y organizados en dos sectores de riego. Las estrategias se han desarrollado con el lenguaje de programación Python y se conectan automáticamente a la red meteorológicas de la AEMET y a la página web de elTiempo.es para obtener las variables meteorológicas. Una vez calculadas las estrategias de riego, se ha procedido al diseño de la aplicación web. La aplicación permite mostrar los datos obtenidos de forma gráfica y modificar las variables relacionadas con el sistema hidráulico, el cultivo y la explotación. De esta manera, usando las ecuaciones para el cálculo de la programación de riego junto con las tecnologías de desarrollo web, se consigue crear una aplicación que presenta una interfaz amigable, para la visualización de los datos permitiendo personalizar las variables relacionadas con la programación del riego, de forma que se puede adaptar a cualquier explotación.

Palabras Clave:

Riego de precisión, Cultivo del Almendro, TIC’s, Software, Aplicación Web

Abstract:

This work aims to develop a methodology for the implementation of a web application that calculates a precision irrigation. Initially, the work designs the irrigation schedule of the study farm. This farm has a total area of 44 ha of intensive almond trees with two irrigation sectors. The strategies have been developed with Python programming language and are automatically connected to the AEMET meteorological network and to the elTiempo.es website to obtain the meteorological variables. Once the irrigation strategies have been calculated, the web application has been designed. The application allows to display the data obtained graphically and to modify the variables related to the hydraulic system, the crop system and farm conditions. In this way, by the equations to calculate the irrigation schedule together with web development technologies, it is possible to create an application that presents a friendly interface for data visualization, and allows modifying the variables related to irrigation, so that it can be adapted to any farm.

Keywords :

Precision Irrigation, Almond, TIC's, Software, Web Application

Tabla de contenido

Lista de figuras:

Lista de tablas:

Acrónimos:

OCDE: La Organización para la Cooperación y el Desarrollo Económico

INE: Instituto Nacional de Estadísticas

RP: Riego de precisión

TICs: Tecnologías de Información y Comunicación

NDVI: *Normalized Difference Vegetation Index*

TFM: Trabajo Final de Máster

JS: *JavaScript*

GPRS: *General Packet Radio Service*

MVC: Modelo Vista Controlador

FAO: *Food and Agriculture Organization*

API REST: *Application Programming Interface. REpresentational State Transfer*

RDC: Riego Deficitario Controlado

AFA: Agua Fácilmente Aprovechable

ADT: Agua Disponible Total

SPA: *Single Page Application*

JSX: *JavaScript XML*

HTTP: *Hypertext Transfer Protocol*

JSON: *JavaScript Object Notation*

SQL: *Structured Query Language*

CSRF: *Cross-Site Request Forgery*

DRF: *Django Rest Framework*

DOM: *Document Object Model*

AJAX: *Asynchronous JavaScript And XML*

WSGI: *Web Server Gateway Interface*

XSS: *Cross Site Scripting*

Px: *Pixels*

Capítulo 1 - Memoria

1. Introducción

Según estimaciones de la Organización para la Cooperación y el Desarrollo Económico (OCDE), para 2050 más del 40% de la población vivirá bajo un grave estrés hídrico a medida que la demanda de agua aumente un 55%. La agricultura y la ganadería son los principales usuarios del agua en España con un 82% del total del agua consumida, según datos del Instituto Nacional de Estadísticas (INE) en 2018. Un informe realizado por el Ministerio de Agricultura y Pesca en 2018 estima que la superficie de regadío en España es de 3.774.226 ha (22.18% del total), lo que supone un aumento del 3,25% desde 2016. Andalucía es la comunidad autónoma que más porcentaje de superficie regada presenta, con un 29,2% del total.

En España, al igual que ocurre en el resto de los países de la cuenca mediterránea, el agua es un recurso escaso, frágil e irregularmente distribuido. A la escasez de precipitaciones, la elevada evapotranspiración, la alta variabilidad espacial y temporal de las lluvias, la desigual distribución de los recursos hídricos y la frecuencia de las sequias, se unen problemas como son el aumento de la presión sobre los recursos hídricos, el déficit creciente que sufren algunas cuencas y la salinización o contaminación de acuíferos (García et al, 2015).

En la actualidad, y más aún en el futuro, la agricultura de regadío se llevará a cabo bajo un escenario de escasez de agua. El suministro insuficiente de agua para el riego será la norma más que la excepción, y la gestión del riego pasará de enfatizar la producción por unidad de área, hacia la maximización de la producción por unidad de agua consumida (Fereres & Soriano, 2007).

Debido a que la agricultura de regadío es uno de los principales consumidores de agua, un bien cada vez más escaso, se ha visto una tendencia a la implantación de riego localizado, lo que permite un gran ahorro de este recurso y un control mucho más efectivo del riego. Sin embargo, el uso de este sistema por sí solo no es suficiente para solventar los problemas actuales y futuros en el regadío. Es por esto por lo que cada vez es más necesario la implantación de estrategias de riego deficitario y de nuevos métodos de programación de riego. El conjunto de estas estrategias, métodos y sistemas de riego más eficientes es lo que se define como Riego de Precisión (RP).

El RP, impulsado por el desarrollo tecnológico, que implica la reducción del coste de los sensores, supone un cambio importante con respecto a la concepción tradicional de la agricultura de regadío. Este tipo de riego tiene un mayor control sobre los distintos factores que intervienen en el mismo, mejorando la gestión en el uso del agua. Esto permite la reducción de costes; la mejora en la aplicación del riego y la disminución del impacto medioambiental, lo que hace cada vez más imprescindible su uso en la agricultura.

Para realizar un buen riego de precisión se pueden utilizar un conjunto de herramientas denominado Tecnologías de Información y Comunicación (TICs), que permiten la trasmisión, el procesamiento y el almacenamiento de información de forma digital. Con su uso, se posibilita el manejo de factores claves en el riego tales como el estado del sistema de riego, valores climáticos, valores de humedad de suelo, así como cualquier otra información útil. Además, posibilitan la automatización de diferentes tareas de gestión haciendo uso de dichos factores.

El presente “Trabajo Final de Máster” desarrolla una aplicación web para la automatización de un sistema de riego en una finca de almendros situada en Alcolea (Córdoba), junto al río Guadalquivir.

2. Justificación de la necesidad del presente trabajo

El presente proyecto nace de la necesidad, por parte de los agricultores, de implantar de forma sencilla y eficaz un riego de precisión. Puesto que generalmente los sensores poseen plataformas independientes, es necesario desarrollar herramientas que integren toda la información y ayuden al agricultor en la toma de decisiones, mostrándole los aspectos claves para llevar acabo un RP. Además, se generarán distintas estrategias de riego, basadas en modelos predictivos, que van a informar al agricultor sobre las necesidades hídricas de su cultivo a lo largo de la semana, mejorando así la toma de decisiones.

En el aspecto técnico del proyecto se pretende desarrollar una aplicación web para el manejo del riego, usando información de sensores y otras fuentes de datos. Se incluirán distintas estrategias de riego, para que el agricultor pueda usar la que mejor se adapte a sus condiciones. Además, la aplicación web mostrará en tiempo real los datos medidos por los sensores, las predicciones meteorológicas, el estado del sistema de riego, las características de la explotación y mapas de índice de vegetación de diferencia normalizada (NDVI).

Capítulo 2 - Objetivos

El objetivo principal de este Trabajo Final de Máster (TFM) es realizar una aplicación web que permita la implantación de un riego de precisión para el cultivo del almendro.

El objetivo principal se puede dividir en los siguientes objetivos secundarios:

- Definición e integración en la herramienta de estrategias de riego para el cultivo del almendro.
- Automatizar el riego de precisión.
- Algoritmo de conexión con la nube para la obtención de las variables climáticas necesarias para llevar a cabo la programación del riego de precisión.
- Desarrollo de herramientas para el análisis gráfico de las estrategias de riego diseñadas.
- Permitir que el usuario pueda analizar diversas estrategias de riego y aplicar las más conveniente para su explotación.
- Creación de una aplicación web versátil que pueda adaptarse a distintos cultivos y que pueda servir de base para futuros proyectos.

Capítulo 3 - Materiales y métodos

1. Área de estudio

El proyecto se ha llevado a cabo en una finca comercial de almendros situada en Alcolea (Córdoba), junto al rio Guadalquivir (latitud 37,9368, longitud -4,6553)(Figura 1).

Figura 1 Ubicación de la finca. Fuente: Elaboración propia

El clima en esta zona se caracteriza por inviernos suaves con temperaturas que rondan los 10ºC y veranos calurosos con temperaturas medias superiores a los 27ºC (Tabla 1).

Tabla 1 Medias de los datos históricos recogidos en la estación meteorológica de Córdoba desde el año 1994 hasta el 2019, obtenidos haciendo uso de la API de AEMET. Fuente: elaboración propia

Meses	Precip (mm)	Tª Max (ºC)	Tª Min (ºC)	V. Viento (Km/h)	Insolación (h)	ET0 (mm)
Enero	63,97	15,07	4,01	6,8	5,64	1,16
Febrero	52,52	17,44	4,97	6,96	6,57	2,32
Marzo	64,81	21	7,47	8,2	7,2	3,45
Abril	53,42	23,84	10	9,36	8,12	4,59
Mayo	44,35	28,18	13,03	9,16	9,7	5,8
Junio	7,8	33,61	16,94	9,64	11,32	6,89
Julio	0,49	36,86	19,12	9,84	12,04	7,27
Agosto	5,62	36,83	19,66	8,84	11,22	6,33
Septiembre	36,71	31,67	17,21	7,68	8,47	3,67
Octubre	75,03	26,34	13,37	6,6	7,2	2,04
Noviembre	83,4	19,09	7,83	6,4	5,98	0,81
Diciembre	104,22	15,74	5,27	7,32	5,25	0,59
Año	592,34	25,47	11,58	8	8,23	2,24

Las precipitaciones anuales rondan los 592 mm, pudiendo llegar algunos años a superar los 1.000 mm (Figura 2). El año 2019, fecha en la que se inició el proyecto, fue un año seco con una precipitación anual de 339,5 mm.

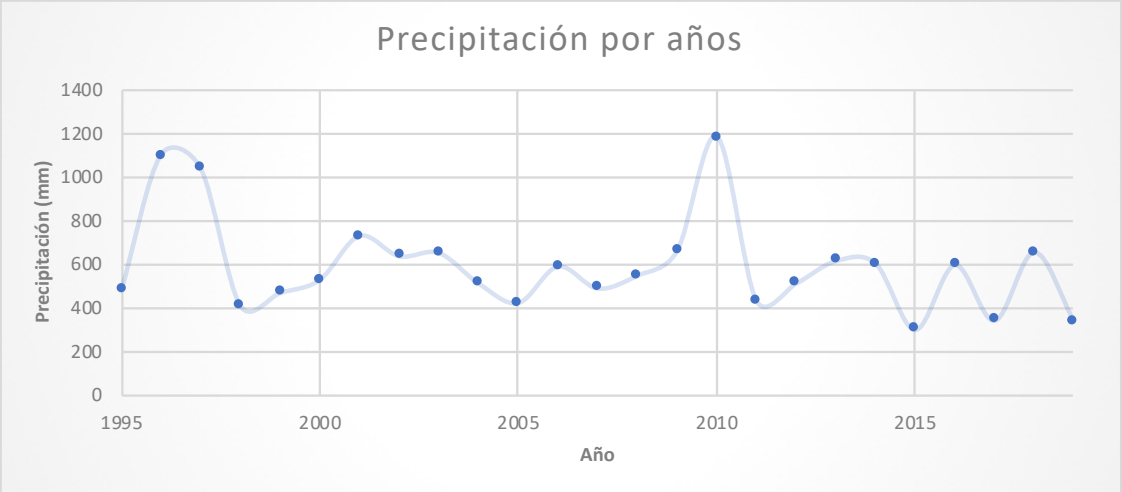

Figura 2 Precipitación según datos históricos recogidos en la estación meteorológica de Córdoba desde el año 1995 hasta el 2019, obtenidos haciendo uso de la API de AEMET. Fuente: Elaboración propia.

La finca presenta una pendiente media de -0,9% medida desde la parte sur hasta la norte (Figura 3). Alcanzando su mayor altitud a 113 metros sobre el nivel del mar y su mínima a 110 metros. La zona con menor altitud se localiza junto al margen del rio, mientras que las zonas de más altitud se sitúan en el interior de la parcela. No obstante, la finca presenta una pendiente mínima y las diferencias de cota son prácticamente nulas.

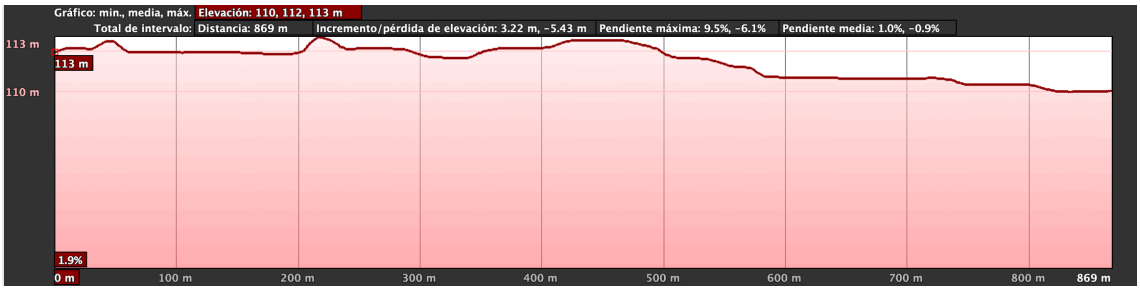

Figura 3 Perfil de elevación de la finca donde se realizo el estudio en dirección Sur-Norte obtenida haciendo uso del software Google Earth Pro. Fuente: Elaboración propia

El terreno presenta una textura que varía entre franco-arcillosa a franco-arenosa en función de la zona.

Dicha finca consta de dos variedades de almendro, Soleta y Lauranne, distribuidas en dos parcelas con una superficie productiva de 22 y 21 hectáreas respectivamente. La plantación es de tipo intensiva con un marco de 6 x 3,5 metros.

El riego está dividido en dos sectores, uno para cada variedad de almendro, y cada uno consta de dos bombas. Una de las bombas se encargará de llevar el agua del rio Guadalquivir hasta un deposito de un millón de litros. La segunda bomba es utilizada para aplicar riego a los distintos sectores. Ambas bombas se alimentan mediante energía fotovoltaica. La finca cuenta también con un sistema de fertirriego, de modo que le aplican los fertilizantes directamente en el agua de riego.

La finca de estudio presenta un riego por goteo, con un caudal de gotero de 1,2 L/h (Figura 4).

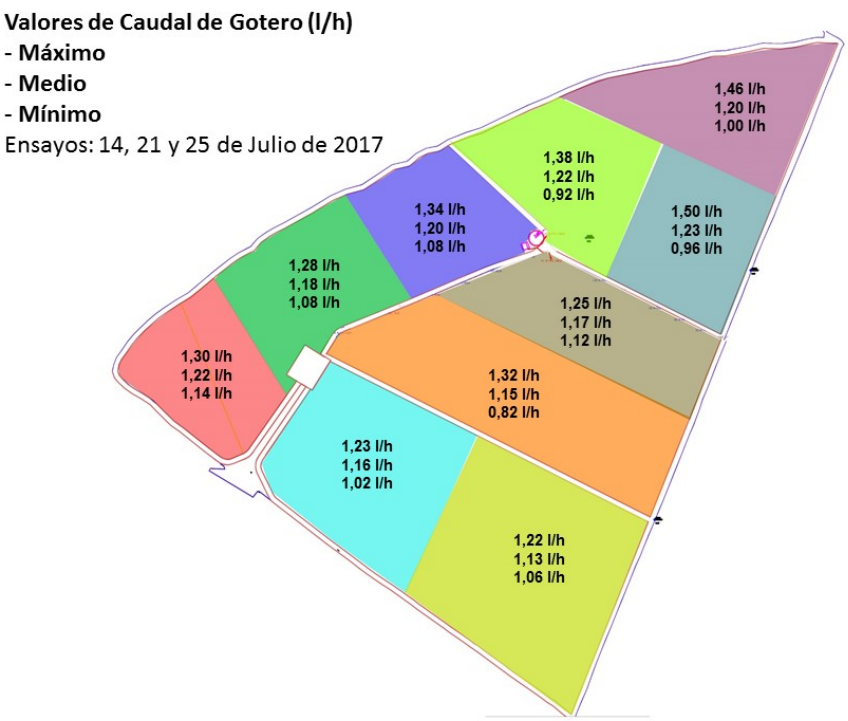

Figura 4 Caudal de los goteros. Fuente: Universidad de Córdoba, elaborado por Carmen Alcaide Zaragoza

El número de goteros se ha obtenido a partir de una representación realizada en el software QGIS (Figura 5)

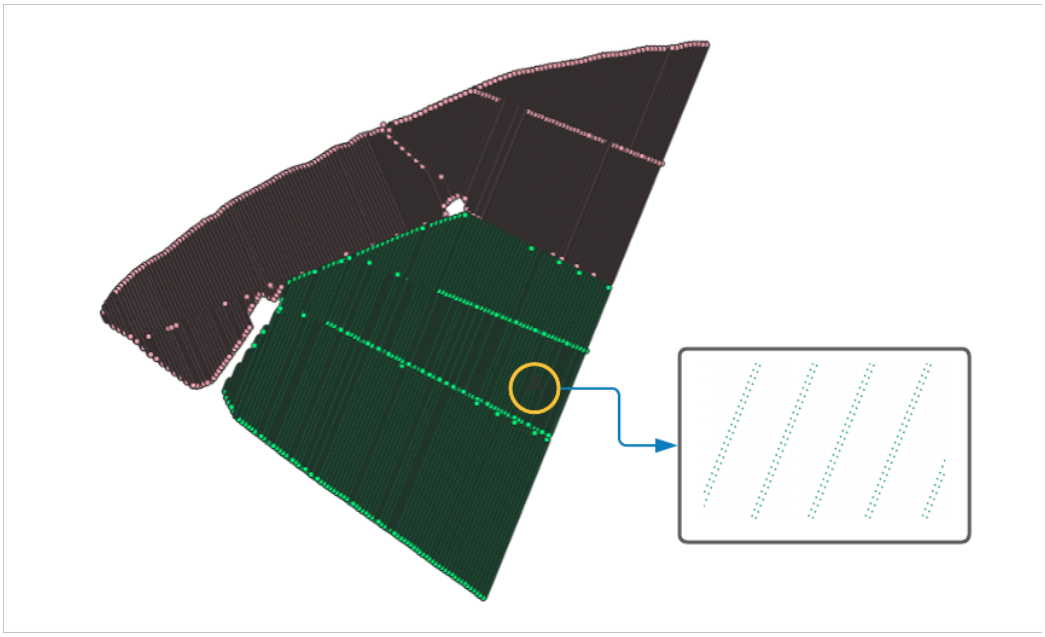

Figura 5 Representación sistema de riego, sector 1: rosa (lauranne), sector 2: verde (soleta). Fuente: Universidad de Córdoba, elaborado por Carmen Alcaide Zaragoza

Las características del sistema de riego por sectores se representan en la Tabla 2 Los valores de uniformidad y eficiencia de riego se han obtenido de un ensayo de campo que realizó la Universidad de Córdoba.

Tabla 2 Características del sistema de riego por sectores. Fuente: elaboración propia

Sector	Nº Goteros	Caudal goteros (l/h)	Uniformidad del riego (%)	Eficiencia del riego (%)
1	87000	1,18	95,5	90
2	93000	1,15	90,4	90

2. Materiales

2.1 Materiales de campo

El cálculo de la humedad en la zona radicular se ha realizado mediante 4 sensores de humedad de suelo de tipo capacitivo instalados en la finca (Figura 6). Los sensores se disponen en grupos de dos, encontrándose cada grupo en un sector de riego diferente.

Figura 6 Sonda de humedad y conductividad eléctrica. Fuente: www.wiseagrotecnologia.com

Para medir la humedad a diferentes niveles de la zona radicular se posiciona en el suelo un sensor a 30 cm y otro a 50 cm de profundidad. El sensor más superficial va a mostrar el contenido de humedad en la primera capa del terreno, siendo el primero en experimentar cambios de humedad producidos a consecuencia de un riego o lluvia. El otro sensor muestra la humedad en la zona más profunda de las raíces. Valores altos de humedad en el sensor de más profundidad se traducen en pérdidas de agua por percolación.

Los sensores se conectan por radio frecuencia a un dispositivo denominado *Gateway*, que se encuentra en el interior de la caseta de riego, cuyo propósito es traducir la información que le llega de los sensores, para que sea compatible con el protocolo utilizado en el servidor web, y enviarla vía GPRS (General Packet Radio Service), a una base de datos propiedad de la empresa suministradora de dichos sensores, para posteriormente mostrarlos en su web.

2.2 Lenguajes de programación

En cuanto al software, para la realización del trabajo ha sido imprescindible el uso de las siguientes tecnologías:

Como entorno de desarrollo se ha utilizado el editor de código Visual Studio Code versión 1.49.0. Este dispone de soporte para un gran número de lenguajes de programación y de un sistema de paquetes que amplia notablemente sus posibilidades, ayudando así al desarrollo del proyecto. El lenguaje principal de programación usado en este proyecto es Python versión 3.7.7.

Para el desarrollo de la página web se han utilizado los siguientes *frameworks*:

- Django en su versión 3.0.2 para el desarrollo del *back-end*. Este *framework* es de código abierto, está escrito en Python y sigue un patrón de diseño conocido como Modelo-Vista-Controlador (MVC)

- React en su versión 16.13.1 para el desarrollo del *front-end*. Este *framework* es de código abierto, desarrollado por Facebook, está escrito en JavaScript (JS) y representa la vista en el patrón MVC.

La arquitectura MVC es un patrón de diseño de software que separa en tres componentes los datos, la metodología y la interfaz gráfica. El modelo se encarga de gestionar, manipular y actualizar la base de datos. La vista es el componente encargado de mostrarle al usuario la parte visual, es decir, las pantallas, formularios, ventanas, etc. El controlador es el componente principal ya que se encarga de la lógica de la aplicación, gestionando las instrucciones que se reciben y procesándolas. A través del controlador, se realizan las diferentes consultas al modelo y una vez obtenidos los datos los envía a la vista para que los muestre al usuario.

Para guardar los datos tanto del sistema de riego como de los sensores de campo se ha utilizado una base de datos relacional gestionada mediante MySQL. Finalmente se ha integrado toda la aplicación dentro de Docker, que permite la automatización y el despliegue de la aplicación en diferentes dispositivos.

2.3 Hardware

Para la puesta en producción se ha usado una Raspberry Pi 3 Model B+ como servidor (Figura 7). La Raspberry es un ordenador de placa reducida, al que se le pueden conectar periféricos de entrada y salida para poder interactuar, como un ratón, un teclado y una pantalla.

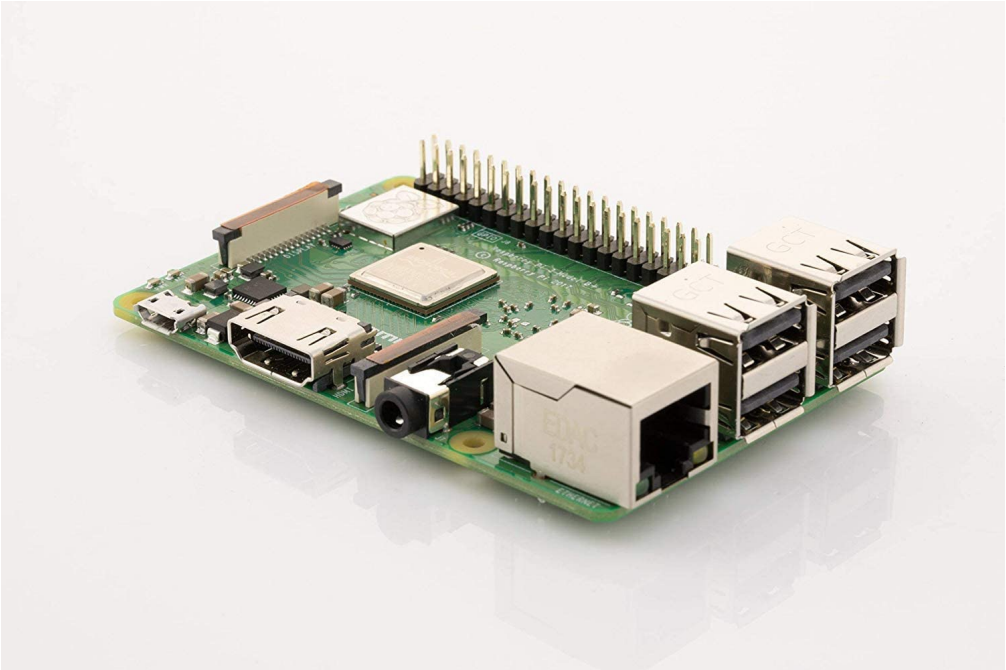

Figura 7 Raspberry Pi 3 B+. Fuente: www.www.raspberrypi.org

La tabla 3 muestra las características de la Raspberry Pi utilizada en este proyecto.

Tabla 3 Características Raspberry Pi 3 B+. Fuente: elaboración propia

RASPBERRY PI 3 MODEL B+	
CPU + GPU	Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
RAM	1GB LPDDR2 SDRAM
Wi-Fi + Bluetooth	2.4GHz y 5GHz IEEE 802.11.b/g/n/ac, Bluetooth 4.2, BLE
Ethernet	Gigabit Ethernet sobre USB 2.0 (300 Mbps)
Puertos	1 HDMI, 4 USB 2.0 , CSI, DSI Micro-SD, Power-over-Ethernet (PoE), audio estéreo y video compuesto

3. Metodología.

La metodología se puede dividir en dos partes principales, el desarrollo de la aplicación web y el cálculo de la programación de riego. Ambas partes están relacionadas, de forma que una depende de la otra según se muestra en la Figura 8.

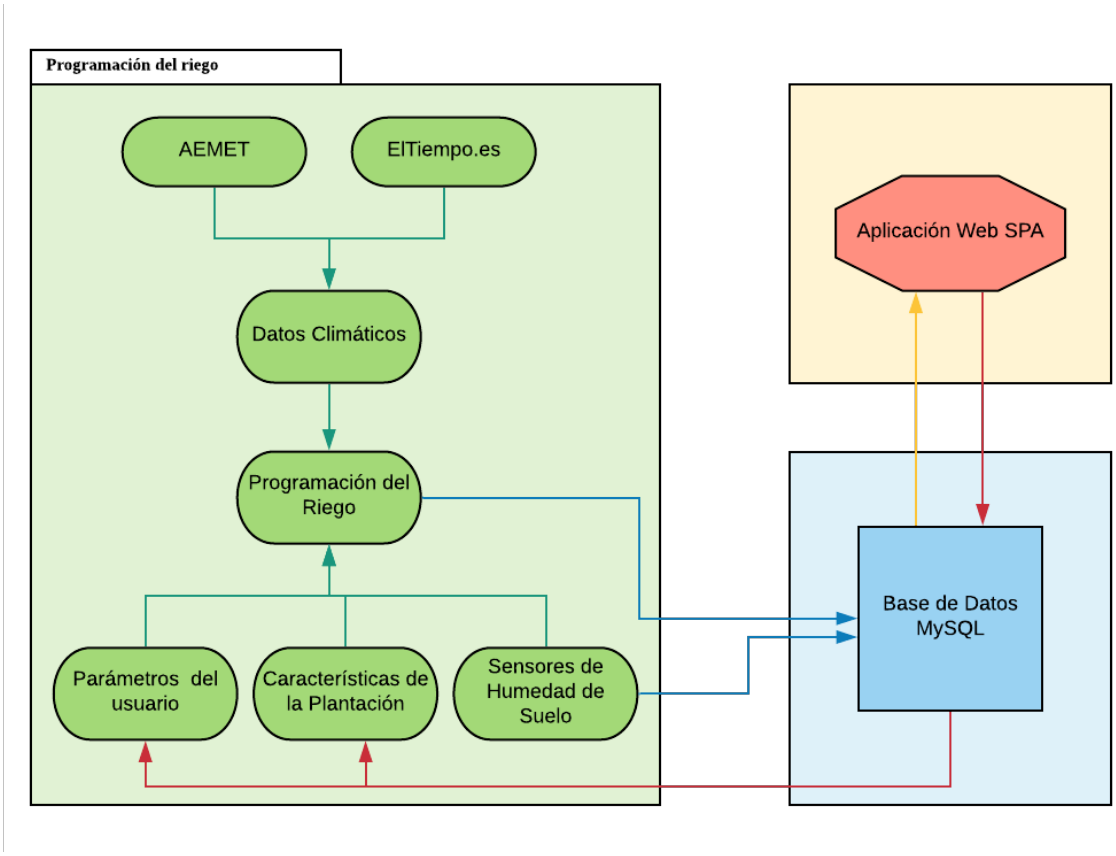

Figura 8 Esquema general del funcionamiento de la aplicación. Fuente: Elaboración propia

Como se observa en la Figura 8 la programación de riego y la aplicación web se comunican a través de la base de datos. Todos se encuentran alojados de manera independiente en una Raspberry Pi. El funcionamiento es el siguiente:

- El usuario se conecta a la aplicación y rellena un formulario en el que definirá los parámetros necesarios para el cálculo de la programación de riego: características del sistema de riego, características de la explotación y el sector, y parámetros definidos por el usuario (tiempo de riego máximo, dotación, agotamiento de humedad permitido, etc.). Esto permite que la aplicación sea dinámica y se adapte a cualquier explotación. Una vez rellenado el formulario, la aplicación web almacena los datos en la base de datos.

- A su vez un algoritmo se encarga de rescatar, mediante Web Scraping, los valores de los sensores que se encuentran alojados en la web de la empresa suministradora. Una vez obtenidos los valores, los almacena en la base de datos. Este proceso se repite cada 15 minutos, que es el tiempo de espera del sensor.

- Tras esto, la programación de riego se conecta con la base de datos para obtener los parámetros definidos por el usuario y el valor de humedad del sensor, y calcula el tiempo de riego, en función de la estrategia. Tras su cálculo almacena los datos en la base de datos

- Por ultimo la aplicación web accede a la base de datos, cuando el usuario se conecta, y muestra de forma gráfica los valores de los sensores y los resultados obtenidos en la programación de riego

Para la programación del riego de precisión se ha utilizado la metodología propuesta por la Organización de las Naciones Unidas para la Agricultura y la Alimentación (FAO) (Allen et al., 1998). Estas ecuaciones se han implementado en el servidor (Raspberry Pi) de forma que permiten el cálculo automático diario del riego.

El desarrollo de la aplicación web se divide en tres partes, el *back-end*, el *front-end* y el despliegue. El primero hace referencia a todos los procesos que permiten que la aplicación web funcione correctamente, como es el procesamiento de los datos, el cálculo de las estrategias de riego, la interacción con la base de datos, etc. Este va a seguir la arquitectura MVC, permitiendo que la aplicación sea modular y altamente escalable. El segundo se enfoca en el elemento gráfico de la web (interfaz de usuario), es decir, en todo lo que el usuario puede ver e interactuar mientras navega. Este va a utilizar una extensión de JS denominada JSX que combina los lenguajes de programación HTML5 (encargado de la estructura), CSS3 (encargado del estilo) y JS (encargado de la lógica). Por último, el despliegue consiste en instalar la aplicación en una Raspberry Pi para que esté activa en todo momento.

Se ha buscado en todo momento facilitar la integración del *back-end* con el *front-end,* utilizando Python como lenguaje de programación para todo el *back-end* y JS para todo el *front-end*. El desarrollo e implementación tanto del *back-end* como del *front-end* se explicarán de forma detallada en el capítulo 3.

Finalmente, para aislar la programación de riego, la base de datos y el servidor web, de forma que trabajen de forma totalmente independientes, es necesario el uso de una herramienta denominada Docker.

Docker se puede definir de forma técnica como un software libre y de código abierto que posibilita el despliegue de aplicaciones dentro de contenedores. Básicamente lo que permite Docker es desplegar una aplicación en cualquier dispositivo independientemente del sistema operativo que este tenga, evitando así incompatibilidades. Además, permite aislar distintas partes de la aplicación de modo que, si alguna presenta algún error, las demás seguirán funcionando sin ningún problema.

3.1 Programación del riego

Lo primero a la hora de programar el riego es escoger la estrategia que más se ajuste a las necesidades del usuario. La elección de la estrategia va a depender de diversos factores como el tipo de cultivo, el sistema de riego, la disponibilidad de agua, el coste de explotación de esta y el precio final del producto. Conociendo estos factores el agricultor debe elegir la estrategia que le aporte el máximo beneficio, que por lo general es distinta a la seguida para alcanzar la máxima producción (Figura 9).

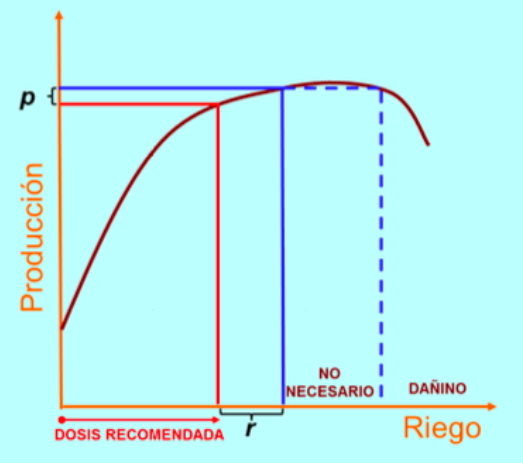

Figura 9 Respuesta de la producción de un cultivo al riego. Fuente: IRNAS, 2015

En la aplicación se puede escoger entre dos tipos de estrategias: 1- Reponer 100% las necesidades de agua, 2- Riego deficitario controlado (RDC). La Figura 10 muestra un esquema del cálculo de la programación del riego en función de la estrategia escogida.

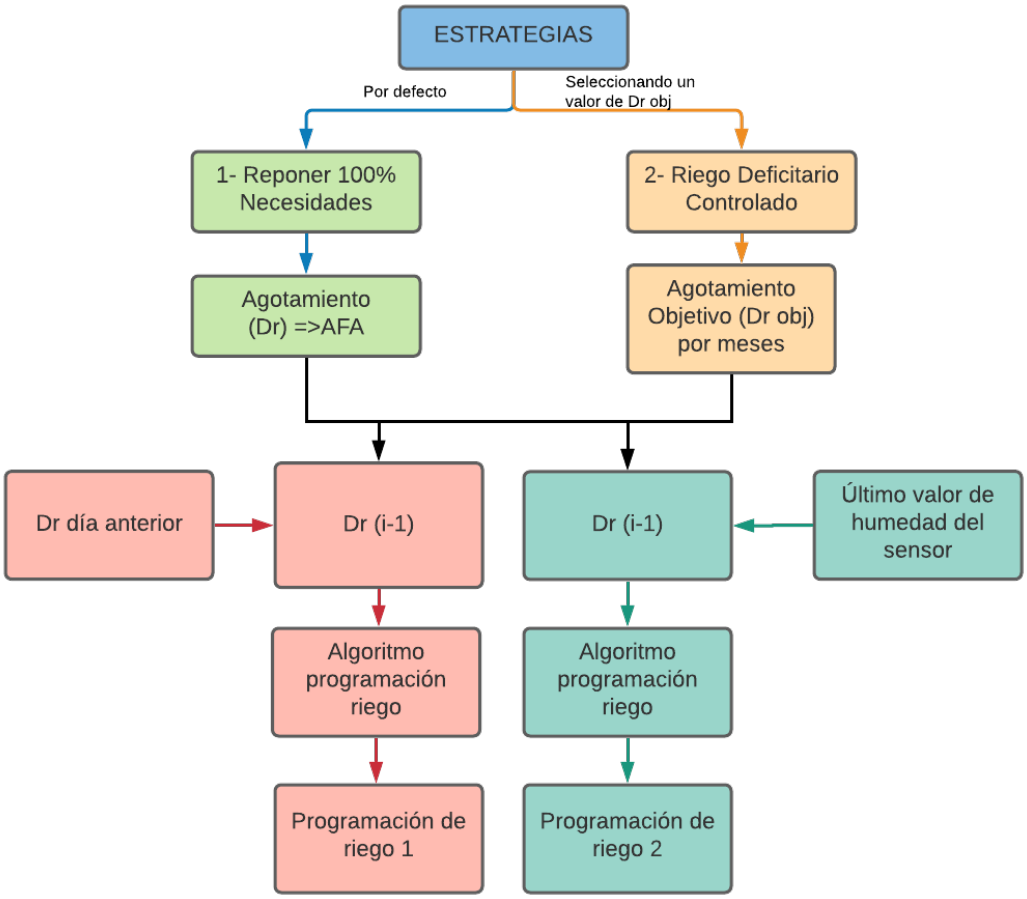

Figura 10 Esquema de la programación del riego para las diferentes estrategias. Fuente: elaboración propia

Como se observa en el esquema de la Figura 10, la programación de riego por defecto utiliza la estrategia de reponer 100% las necesidades y calcula el tiempo de riego que el sistema debe estar funcionando para que el agotamiento de humedad en el suelo sea igual o menor al Agua Fácilmente Aprovechable (AFA). Este proceso lo repite para cada día de los próximos siete. De esta forma, aplicando el algoritmo utilizado para la programación del riego, que se explicará en el siguiente apartado, se obtiene la lámina bruta y el tiempo de riego necesarios para que el suelo no presente agotamiento de humedad.

Si se quisiera realizar un RDC se asigna un valor de agotamiento objetivo que establece el nivel máximo de agotamiento que el suelo debe presentar. De esta manera se puede reducir los aportes hídricos en aquellos periodos fenológicos en los que aplicar un déficit hídrico controlado no afecte sensiblemente a la producción y a la calidad de la cosecha, mientras que en el resto del ciclo del cultivo se garantice el 100% de las necesidades hídricas.

Una vez decidida la estrategia de riego a utilizar, el sistema las calcula de dos formas diferentes:

- En la primera el valor de humedad del suelo del día anterior $D_{r,i-1}$ se calculará cada lunes a partir del valor que den los sensores. Sin embargo, para los demás días de la semana tomará el valor de $D_{r,i-1}$que se estimó para ese día. Esta estrategia se centra más en los cambios producidos por la predicción meteorológica.

- En la segunda, $D_{r,i-1}$ se calculará todos los días a partir de los valores que presenten los sensores de humedad instalados en la finca. Esta se centra más en los datos que aportan los sensores

3.1.1 Algoritmos para la programación de riego

El riego de precisión se basa en aplicar el agua de riego al cultivo en el momento y cantidad necesaria para su correcto desarrollo. Para ello se deben estudiar las relaciones que existen entre el clima, el cultivo, el suelo, y el sistema de riego.

La traspiración es el proceso de la planta que va a determinar la cantidad de agua que esta necesita. Este proceso va a depender del tipo de cultivo, de la cantidad de agua que la planta dispone y de variables climáticas. Cuando la planta dispone de buenas condiciones, las estomas se abren y la planta transpira consumiendo mucha agua. Sin embargo, en situaciones de estrés la planta cierra sus estomas para protegerse y el consumo de agua se detiene.

El diseño del algoritmo de programación del riego va a seguir el esquema mostrado en la Figura 11.

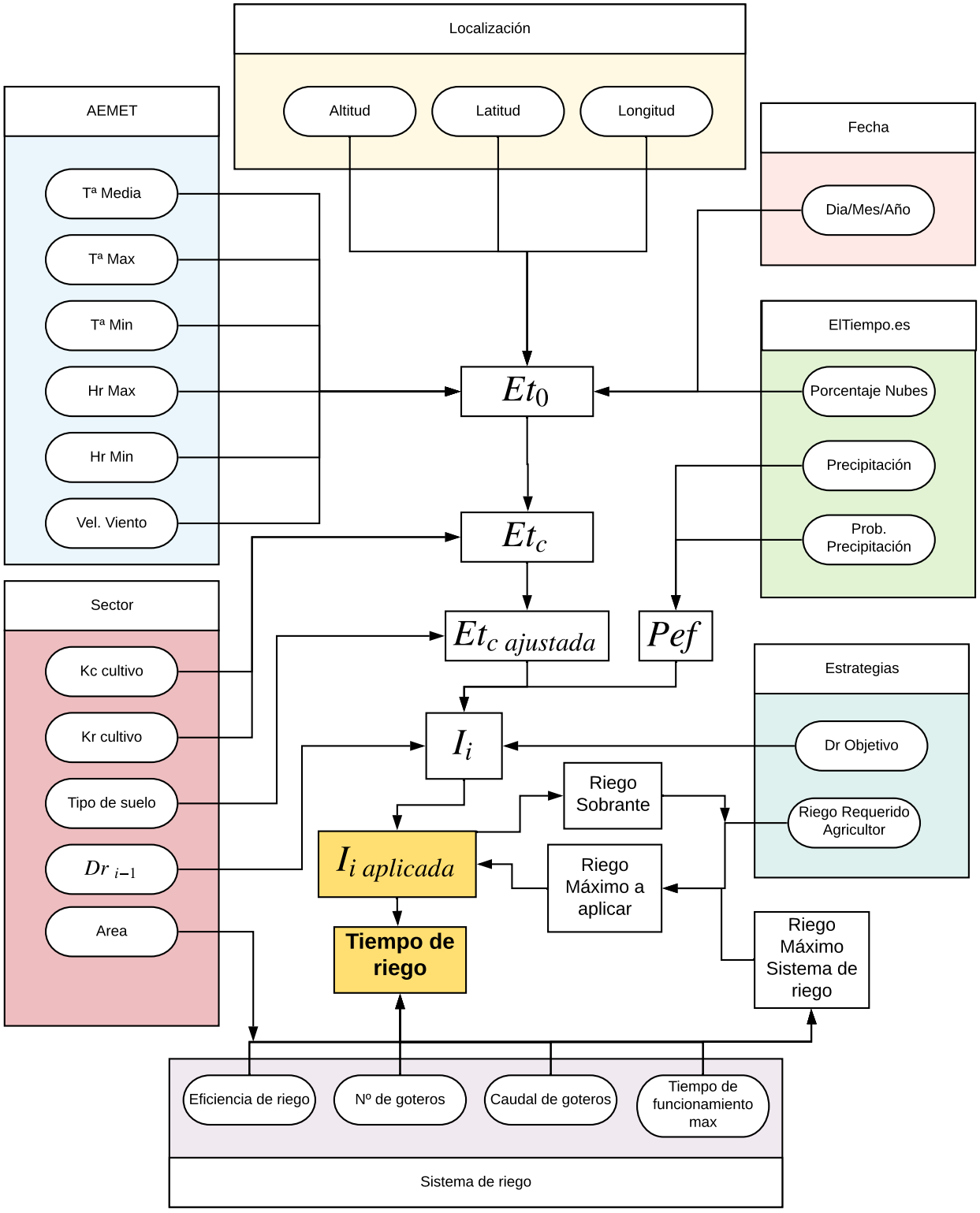

Figura 11 Esquema del diseño de la programación del riego. Fuente: Elaboración propia

En primer lugar, se calcula la evapotranspiración de referencia (ET_0) (mm) que va a depender únicamente de los valores climatológicos de la zona de estudio. A través de la API de la AEMET OpenData se obtienen las variables Tª media (ºC), Tª máxima (ºC), Tª mínima (ºC), humedad relativa máxima (%), humedad relativa mínima (%) y velocidad del viento (m/s). OpenData es una API REST (Application Programming Interface. REpresentational State Transfer) desarrollada por la Agencia Estatal de Meteorología (AEMET) que permite la utilización de la información meteorológica de la agencia. Por otro lado, se obtiene el porcentaje de nubes mediante Web Scraping a la página web de ElTiempo.es.

En segundo lugar, se calcula la evapotranspiración del cultivo (ET_c) (mm) que utiliza el valor del coeficiente del cultivo (Kc) y la ET_0 calculada anteriormente. Este valor tiene que ser ajustado en función del tipo de suelo dando lugar a la evapotranspiración del cultivo ajustada ($ET_{c\ ajustada}$)(mm).

Tras esto se realiza un balance de agua en el suelo para calcular la lámina neta de riego $(I_i)(mm)$, considerando las entradas (precipitación efectiva) y las salidas (evapotranspiración del cultivo.) Para el balance de agua es necesario el agotamiento del suelo del día anterior (Dr_{i-1})(mm), que informa sobre la humedad del suelo, y el agotamiento del suelo al que se quiere llegar ($Dr_{objetivo}$)(mm).

Una vez calculada I_i, se calcula el tiempo que el sistema de riego tiene que estar funcionando. Estas variables van a depender de las limitaciones del sistema hidráulico (Área, eficiencia del riego, nº de goteros, caudal de goteros y tiempo de funcionamiento máximo) y de las limitaciones de agua que el agricultor especifique (riego requerido). Los parámetros referentes a la localización, sistema de riego y a las estrategias, se encuentran almacenados en la base de datos y pueden ser modificados desde la aplicación para ajustar la programación del riego a cualquier sector, cultivo y necesidades del agricultor.

3.1.2 Balance de agua en el Suelo

Como se ha comentado en el apartado anterior, para calcular el tiempo que el sistema de riego tiene que estar funcionando hay que realizar un balance de agua en el suelo (Figura 12) . Este balance se calcula restando los aportes de agua que recibe el suelo, como son el riego, la lluvia y el ascenso capilar del agua subterránea, y sumando los procesos que sustraen dicha agua, como es la evapotranspiración y las pérdidas por percolación y escurrimiento.

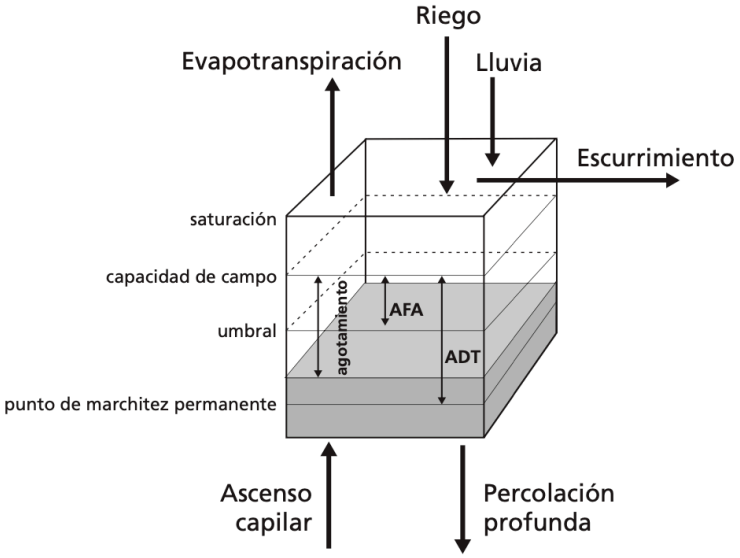

Figura 12 Balance de agua en el suelo. Fuente: FAO, (Allen et al., 1998).

El objetivo del riego de precisión va a ser establecer un nivel objetivo de agotamiento de agua ($D_{r\,obj}$), que debe estar dentro del intervalo de AFA ($D_{r\,obj} < AFA$), de manera que se hagan balances de agua en el suelo para cada día y se aporte la lámina de riego necesaria para mantener dicho nivel de agotamiento.

El balance diario del agua en la zona radicular del suelo, D_r, expresado en términos de agotamiento al final del día, se muestra en la ecuación 1:

$$D_{r,i} = D_{r,i-1} - (P - RO)_i - I_i - CR_i + ET_{c\,aj,i} + DP_i \qquad (1)$$

Siendo:

- $D_{r,i}$ Agotamiento de humedad en la zona de las raíces para el día i (mm)
- $D_{r,i-1}$ Agotamiento de humedad en la zona de las raíces para el día anterior (mm)
- P_i Precipitación efectiva para el día i (mm)
- RO_i Escurrimiento superficial en el día i (mm)
- I_i Lámina neta de riego para el día i (mm)
- CR_i Ascenso capilar en el día i (mm)
- $ET_{c,i}$ Evapotranspiración del cultivo (mm)
- DP_i Pérdidas por percolación profunda en el día i (mm)

El objetivo de la programación del riego es despejar la lámina neta de la ecuación anterior para conseguir un balance de agua que se adapte a la estrategia elegida.

Como ya se ha comentado, se realizarán dos programaciones de riego en función de la forma de cálculo de $D_{r,i-1}$. Para ello es necesario saber el agotamiento del suelo del día anterior y este se puede obtener a partir de dos procedimientos:

- Utilizando el valor de D_r obtenido al realizar el balance de agua del día anterior
- A partir de sensores que midan la humedad de suelo en la zona de estudio

En el caso utilizar el segundo procedimiento es necesario ajustar el valor aportado por los sensores, que suele ser en % de humedad de suelo, a partir de la ecuación 2.

$$D_{r,i-1} = 1000 * \left(\theta_{FC} - \frac{\theta_{i-1}}{100}\right) * Zr \qquad (2)$$

Siendo:

θ_{FC}	Contenido de humedad a capacidad de campo (m^3/m^3)
θ_{i-1}	Contenido de humedad en el suelo (%)
Zr	Profundidad de las raíces (m)

Una vez calculado el $D_{r,i-1}$ hay que calcular las pérdidas de agua que se producen a consecuencia de la evapotranspiración del cultivo. Esta va a depender en primer lugar de la climatología, que está representada por la ET_0. La ET_0 corresponde a la evapotranspiración de una pradera vegetal que cubre totalmente el suelo y crece sin limitaciones de agua y nutrientes, con una altura de unos 10-15 cm. Actualmente la ecuación de Penman-Monteith (Ecuación 3) es la más utilizada para su cálculo (**Allen et al. (1998)).**

$$ET_0 = \frac{0{,}408 * \Delta * (R_n - G) + \gamma * \frac{900}{T+273} * u_2 * (e_s - e_a)}{\Delta + \gamma * (1 + 0{,}34 * u_2)} \qquad (3)$$

Siendo:

ET_0	Evapotraspiración de referencia (mm/día)
R_n	Radiación neta en la superficie del cultivo(MJ/m^2*día)
G	Flujo de calor de suelo (MJ/m^2*día)
T	Temperatura media del aire a 2 metros de altura (ºC)
u_2	Velocidad del viento a 2 metros de altura (m/s)
e_s	Presión de vapor de saturación (kPa)
e_a	Presión real de vapor (Kpa)
e_s-e_a	Déficit de presión de vapor (Kpa)
Δ	Pendiente de la curva de presión de vapor (Kpa/ºC)
γ	Constante psicrométrica (kPa/ºC)

Una vez se obtiene el valor de ET_0, hay que calcular la ET_C. Existen 4 parámetros que van a diferenciar al cultivo utilizado, en este caso almendro, del cultivo de referencia:

- **Altura del cultivo**: Que influye tanto en las características aerodinámicas, como en la trasferencia turbulenta del vapor desde el cultivo hasta la atmosfera
- **Albedo:** porcentaje de radiación que se refleja respecto a la radiación que incide sobre ella, tanto del suelo como del cultivo.
- **Resistencia del cultivo a la transferencia de vapor**: Esta varía en función del área foliar, condición de las hojas, edad y el grado de control estomático.
- **Evaporación de agua en el suelo**.

La ET_C (Ecuación 4) se puede calcular mediante el coeficiente de cultivo (Kc) y un coeficiente de reducción (K_r), a partir de la siguiente ecuación:

$$ET_C = ET_0 * K_c * K_r \quad (4)$$

K_c va a depender de la fisiología de la planta y de la fase de desarrollo en la que se encuentre el cultivo, obteniéndose experimentalmente. Hay que tener en cuenta que este valor también depende de las características de la finca y del manejo. Esto hace que, a la hora de extrapolar el valor de K_c de una finca de referencia a otra, se deba tener en cuenta que este va a ser ligeramente diferente.

La FAO proporciona unas tablas con valores de K_c para una gran cantidad de cultivos. Los valores de K_c para almendro se dividen en las 3 etapas siguientes:

- Etapa inicial: de marzo a abril, K_c = 0,40
- Etapa de mediados de temporada: de mayo a mediados de septiembre, K_c = 0,90
- Etapa final: de mediados de septiembre a octubre, K_c= 0,65

También se ha consultado un estudio del coeficiente de cultivos en Almendro realizado por la junta de Andalucía en la provincia de Córdoba (García y Navarro, 2015) y cuyo resultado fue el que se muestra en la Tabla 4 .

Tabla 4 Evolución quincenal de los coeficientes de cultivo para almendros jóvenes a lo largo del periodo de riego. Fuente: García y Navarro, 2015

Abril	Mayo	Junio	Julio	Agosto	Septiembre	Octubre
0,5-0,6	0,8-1	1-1,1	1,2-1,1	1,11-1	0,9-0,7	0,7

El coeficiente K_r ajusta el valor de K_c en función del tamaño que presente el árbol. Para fracciones de cobertura del suelo que sean superiores al 60%, el valor de K_r será 1, mientras que para coberturas inferiores al 60% el valor de K_r se calculará a partir del gráfico 4, publicado en el manual número 66 de la FAO.

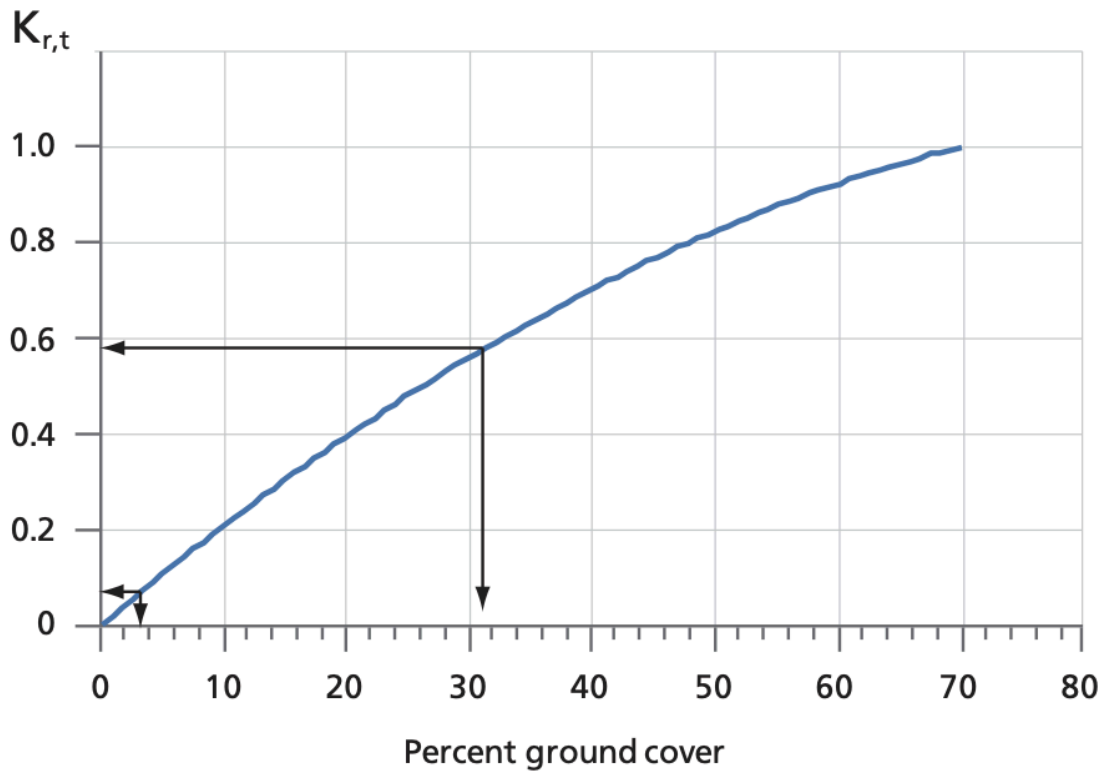

Figura 13 Relación entre la fracción de cobertura de suelo (%) y el coeficiente de reducción Kr Fuente: http://www.fao.org/3/i2800e/i2800e.pdf

Para automatizar el valor de $K_{r,t}$ en coberturas inferiores al 60% se utiliza la ecuación 5:

$$K_r = \frac{2 * S_c}{100} \quad (5)$$

Siendo:

Sc = Cobertura del suelo (%).

Para calcular la cobertura de suelo que presentan los almendros se ha utilizado la ecuación 6:

$$S_c = \frac{\pi * D^2 * N}{400} \quad (6)$$

Siendo:

D = Diámetro medio de copa (m).

N = Densidad de plantación (arboles/ha).

Por otro lado, las raíces de las plantas extraen el agua del suelo con mayor o menor facilidad en función de la humedad que esté presente. Cuando el suelo está húmedo, las partículas del suelo retienen con menor fuerza esa agua y por consiguiente es más fácil su extracción desde la planta. Sin embargo, cuando el suelo está más seco el agua está fuertemente retenida, disminuyendo la capacidad de la planta para extraer el agua disponible. Para ajustar la evapotranspiración del cultivo es necesario saber la cantidad de agua almacenada en el suelo. Para ello la FAO utiliza dos términos adicionales que son el ADT (Agua Disponible Total) y el AFA.

El ADT (Ecuación 7) se refiere a la capacidad de un suelo de retener el agua disponible para las plantas. Este va a depender de la textura de nuestro suelo, que va a condicionar su capacidad de retención, y de la profundidad de las raíces.

$$ADT = 1000 * (\theta_{FC} - \theta_{WP}) * Z_r \quad (7)$$

Siendo:

ADT	Agua disponible total en la zona de las raíces (mm)
θ_{FC}	Contenido de humedad a capacidad de campo (m^3/m^3)
θ_{WP}	Contenido de humedad en el punto de marchitez permanente (m^3/m^3)
Z_r	Profundidad de las raíces (m)

El AFA (Ecuación 8) es la cantidad de agua que un cultivo puede extraer sin mucha dificultad. Cuando el suelo se encuentra próximo a capacidad de campo, es capaz de suministrar el agua suficiente para satisfacer la demanda atmosférica del cultivo. Cuando la humedad en el suelo disminuye en gran medida, el cultivo presentará mayor dificultad para extraer el agua.

$$AFA = p * ADT \quad (8)$$

Siendo:

AFA	Agua fácilmente aprovechable en la zona de las raíces (mm)
p	Fracción del agua total disponible que se puede aguantar sin que el cultivo sufra estrés hídrico

Cuando la cantidad de agua disponible en el suelo es inferior al agua fácilmente aprovechable por la planta (AFA) se produce una reducción de la transpiración del cultivo. La ecuación 9 define la evapotranspiración del cultivo ($ET_{c\,aj}$) en función de la cantidad de agua disponible en el suelo, ADT y AFA.:

$$ET_{c\,aj} = ET_C * \frac{ADT - D_r}{ADT - AFA} \quad (9)$$

Siendo

$ET_{c\,aj}$ Evapotranspiración del cultivo ajustada (mm)

ET_c Evapotranspiración del cultivo. Apartado 3.1.2(mm)

ADT Agua disponible total (mm)

AFA Agua fácilmente aprovechable (mm)

Calculado la $ET_{c\,aj}$ y utilizando los valores previamente definidos de Pe, $D_{r,i-1}$ y D_i, se obtiene la lámina neta de riego necesaria para que el nivel de agotamiento del suelo sea igual al definido en D_i. Sin embargo, la aplicación de la lámina de riego viene limitada por el sistema de riego y la dotación del agricultor, por lo que hay que ajustarla.

El sistema de riego va a ser por lo tanto un factor determinante. Los más comunes son el riego localizado, el riego por aspersión y el riego por gravedad.

Dependiendo del tipo de riego del que se disponga, la eficiencia de aplicación (Ea) va a variar. Esta se puede definir como la relación entre el volumen total de agua aplicado y el almacenado en la zona radicular. En riegos localizados, la eficiencia está en torno al 90%.

Para calcular la lámina de agua que realmente hay que aplicar se define un nuevo término denominado lámina bruta (Ib) que tiene en cuenta el rendimiento del sistema de riego (Ecuación 10).

$$Ib = \frac{I_i}{IE} \quad (10)$$

Siendo:

IE Eficiencia del riego

Una vez que se calcula la lámina bruta que se ha de aplicar, es necesario conocer las características del sistema de riego para calcular las horas que el sistema debe estar en funcionamiento. Como ya se comentó anteriormente la finca se divide en dos sectores hidráulicamente independientes. Para saber el caudal del sistema de riego de cada sector es necesario saber el caudal de los goteros y el número de goteros

El tiempo que el sistema de riego debe estar en funcionamiento se calcula a partir de la ecuación 11:

$$T_i = \frac{I_{b,i} * As * 10000}{ne * qe} \quad (11)$$

Siendo:

T_i	Tiempo de riego para el día i (horas)
$I_{b,i}$	Lámina bruta de agua para el día i, apartado 3.1.2 (mm)
As	Superficie de riego (hectáreas)
ne	Número de goteros
qe	Caudal de los goteros (l/h)

el sistema calculará el riego máximo que es capaz de aplicar cada día, este va a depender del caudal máximo por hora que es capaz de aplicar el sistema de riego, del tiempo máximo que puede funcionar el sistema de riego al día y de la dotación de agua, que limitará el volumen máximo de agua que se puede aplicar.

Finalmente, el algoritmo almacena en la base de datos el riego sobrante, en el caso de que las necesidades del cultivo sean menores que el caudal máximo que se pueda aplicar ese día, y las añade al caudal máximo que se puede aplicar en los días posteriores.

3.2 Diseño de la aplicación web

Una vez realizado el cálculo de la programación de riego es necesario desarrollar una aplicación web que sirva de interfaz para visualizar y editar cualquier variable relacionada con esta programación de riego. Estos datos se deben mostrar en gráficas, se debe permitir al usuario cambiar de forma simple los parámetros correspondientes al calculo de la programación de riego (sistema de riego, cultivo, localización, Dr objetivo, dotación, etc.) y se debe poder acceder a los mismos desde cualquier dispositivo, en cualquier parte del mundo.

Para poder satisfacer esas necesidades se ha desarrollado una aplicación web. Esta tiene una serie de ventajas que la hace particularmente interesante para este tipo de proyectos:

- Es accesible desde cualquier dispositivo
- Es accesible desde cualquier parte del mundo con conexión a internet
- Son fáciles de usar
- Acceso inmediato: no necesitan ser descargadas, instaladas y configuradas

En el diagrama de la Figura 14 se muestra la arquitectura de la aplicación web desarrollada.

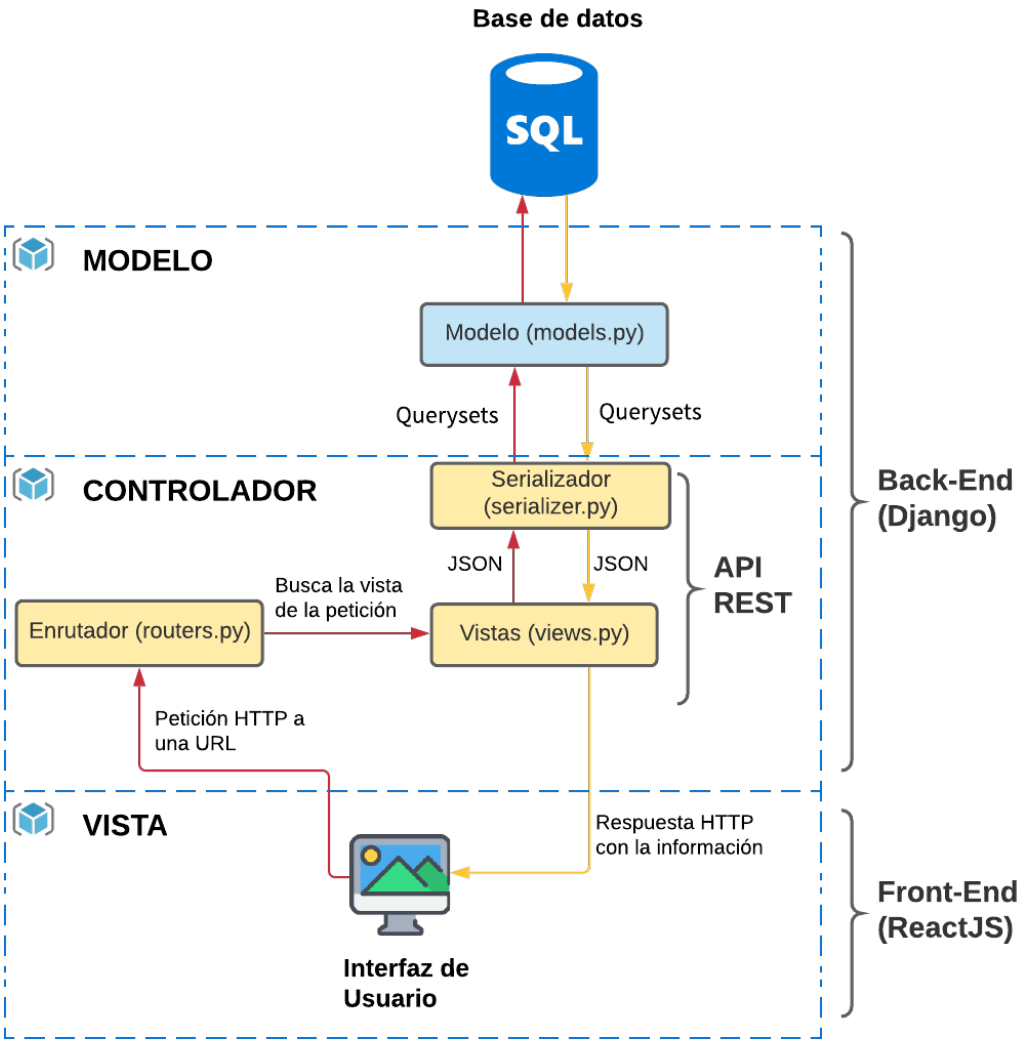

Figura 14 Esquema de la arquitectura MVC de la aplicación web. Fuente: elaboración propia

Como ya se ha comentado antes, el servidor web se va a diseñar usando el *framework* de Django (https://docs.djangoproject.com/) para el *back-end* y se encarga *de* conectar la base de datos con la API-REST. Esta es la parte encargada del correcto funcionamiento de la web, pero no interacciona de manera directa con el usuario, por lo que será el modelo y el controlador en la arquitectura de la aplicación. La elección de este *framework* para el desarrollo del *back-end* del proyecto se basa en el hecho de que está escrito en Python, el mismo lenguaje de programación que se ha utilizado para el cálculo de la programación de riego. El uso de Django aporta una serie de ventajas que van a permitir desarrollar la aplicación de manera profesional. Algunas de esas ventajas son:

- Agilizar el desarrollo
- Facilitar la compatibilidad con los diversos navegadores
- Reducir los errores y ampliar las funcionalidades
- Mejorar notablemente la seguridad

Para la parte del *front-end* se ha utilizado React que es una librería de JavaScript que se utiliza para crear interfaces de usuario, es la parte visible de la web, la vista en la arquitectura MVC, y con la que el usuario interactúa. La elección de esta librería se basa en que es la más utilizada para desarrollo de SPA, por lo que hay una gran comunidad detrás. Además, presenta la ventaja de que funciona mediante componentes, los cuales son reutilizables, lo que facilita el desarrollo del *front-end*. Puesto que el *front-end* y el *back-end* son completamente independientes y están escritos en lenguajes de programación diferentes, necesitan de una API REST para poder comunicarse. Esta API se ha desarrollado dentro del *back-end*. Finalmente, es necesario una base de datos que almacene toda la información

3.2.1 Back-End

El *back-end* de la aplicación será el encargado de gestionar toda la información para que el *front-end* pueda ejecutarse correctamente. Para ello, puesto que ambos están completamente separados, se necesita una API para poder integrarlos.

La API se encarga de conectar los dos sistemas basándose en el protocolo HTTP (Hypertext Transfer Protocol). Con ella se realizan todas las operaciones de consulta, actualización, guardado y borrado en la base de datos conectando así las peticiones realizadas por el usuario (*front-end*) con la gestión de dicha base de datos alojada en *back-end.* Se ha utilizado un formato de texto simple para el intercambio de datos denominado JSON (JavaScript Object Notation)

Como ya se ha comentado el back-end se va a dividir conforme a la arquitectura MVC seguida, donde estaría por una parte el modelo, encargado de interaccionar con la base de datos y el controlador, donde se encuentra la lógica, que corresponde a la API REST.

3.2.1.1 Modelo

El modelo se utiliza para interactuar con la base de datos MySQL. En él se definen las tablas que se van a utilizar y su estructura (Tabla 5). Una vez creadas dichas tablas, se encarga de la conexión con las mismas.

La base de datos cuenta con las siguientes tablas:

Tabla 5 Tablas definidas en la base de datos. Fuente: elaboración propia

TABLA	ALMACENA
Datos AEMET	Todas las variables obtenidas de la AEMET
Datos finca	Datos de la finca (latitud, longitud y altura)
Datos Tiempo.es	La precipitación efectiva y el porcentaje de nubes
Et0	Los valores de ET0
Sectores	Datos de los sectores (Área, terreno, ADT, AFA, etc.)
Coeficiente Kc	Datos de Kc para el cultivo de la finca
ETc Ajustada	Datos de ETc ya ajustados
Agotamiento Dr	Datos sobre el agotamiento de la zona radicular
Riego Requerido	El riego máximo requerido por el usuario
Últimos Datos	Últimos datos aportado por los sensores
Estrategia1 S1	Valores de la estrategia 1 para el sector 1
Estrategias1 S2	Valores de la estrategia 1 para el sector 2
Estrategia2 S1	Valores de la estrategia 2 para el sector 1
Estrategia2 S2	Valores de la estrategia 2 para el sector 2
Sensor Humedad S1	Todos los valores de humedad del sector 1
Sensor Humedad S2	Todos los valores de humedad del sector 2
Alertas S1	Las alertas definidas para el sector 1
Aletas S2	Las alertas definidas para el sector 2
Sensor CE S1	Todos los valores de conductividad del sector 1

Sensor CE S2	Todos los valores de conductividad del sector 2
Número Datos	El número de datos que se muestran en las gráficas
Ajustes Aplicación	Los ajustes de la aplicación web

La base de datos MySQL cuenta con una tabla que permite que la base de datos Redis guarde información de forma persistente en el disco. Esto es importante ya que, si el servidor se cae, la información almacenada en Redis se elimina. Por ello se guardan copias de seguridad, cada cierto tiempo, en MySQL.

3.2.1.2 Diseño de la API REST

El diseño de la API REST se basa en tres componentes esenciales: los serializadores, las vistas y los enrutadores. Estos componentes son los que permiten la conexión del *front-end* con el *back-end.*

Puesto que el *front-end* y el *back-end* están escritos en lenguajes de programación diferentes, se utilizan los serializadores para transformar los datos provenientes de Django (back-end) a formato JSON, para así poder ser utilizados por React (front-end), y permiten hacerlo en ambas direcciones. Básicamente su objetivo es "traducir" el modelo, descrito en el apartado anterior, a formato JSON y viceversa. Además, permiten modificar el valor que llega del *front-end* antes de insertarlo en la base de datos. De esta forma, puesto que los valores de AFA, ADT y Kr se obtienen a partir de otros parámetros (terreno, cultivo y profundidad de las raíces para los dos primeros y diámetro de copa y arboles por ha para el ultimo), se usarán los serializadores para calcularlos e insertarlos en la base de datos cada vez que alguno de estos parámetros se modifica.

Las vistas son el módulo encargado de la lógica, ya que gestiona todas las comunicaciones entre la API y el modelo. Cuando el *front-end* envía una petición, las vistas se encargan de conectarse con los serializadores que a su vez se conectarán con el modelo.

En la Tabla 6, se muestra una lista de las diferentes vistas que van a formar la aplicación:

Tabla 6 Vistas que forman la API REST. Fuente: elaboración propia

Vistas	**Crea una API que:**
Usuario	Muestra los datos del usuario que haya iniciado sesión
Ajustes	Muestra los ajustes de la aplicación del usuario y le permite al *front-end* actualizarlos
Agotamiento Dr	Muestra los datos de agotamiento del suelo para los próximos 7 días
Estrategia 1 Sector 1	Muestra la estrategia primera de riego calculada para el sector 1
Estrategia 1 Sector 2	Muestra la estrategia primera de riego calculada para el sector 2
Estrategia 2 Sector 1	Muestra la estrategia segunda de riego calculada para el sector 1
Estrategia 2 Sector 2	Muestra la estrategia segunda de riego calculada para el sector 2
Coeficiente Kc	Muestra el coeficiente kc por mes para ese cultivo y le permite al *front-end* actualizarlo
Características sector	Muestra todas las características de los sectores necesarias para el calcula de las estrategias y le permite al *front-end* actualizarlas
Riego requerido	Muestra el riego requerido por mes para cada sector y le permite al *front-end* actualizarlo
Número de datos	Muestra el numero de datos que se quiere desplegar en las gráficas y permite al *front-end* actualizarlos
Lauranne Humedad	Muestra los datos de humedad del sensor Lauranne y los filtra en función del numero de datos asignados en los ajustes de la aplicación
Soleta Humedad	Muestra los datos de humedad del sensor Soleta y los filtra
Lauranne CE	Muestra los datos de conductividad eléctrica del sensor Lauranne y los filtra
Soleta CE	Muestra los datos de conductividad eléctrica del sensor Soleta y los filtra
Últimos Valores	Muestra los últimos valores captados por los sensores.

Los enrutadores o routers se encargan de definir las URLs de la API. Hay un enrutador por vista, el cual toma como parámetros el nombre de la vista y la URL de la API creada. Cuando el *front-end* le hace una petición a una URL, el enrutador sabe a qué vista pertenece y le envía dicha petición con el método HTTP utilizado (GET, POST , PUT ...).

3.2.2 Front-end

El *front-end* se puede definir como las tecnologías de diseño y desarrollo web que permiten que el usuario interactúe con la información digital y es conocido como "el lado del cliente". El navegador, con el uso de esta tecnología, transforma la información que le llega del servidor en una interfaz de usuario.

Para el desarrollo del *front-end* se han utilizado las plantillas de Django para la gestión de usuarios y React para el resto de la aplicación web. Esta librería permite crear aplicaciones web de una sola página (SPA), aumentando significativamente la experiencia del usuario y el rendimiento de la aplicación. La aplicación web consta realmente de un solo documento HTML y conforme el usuario interacciona, el contenido va cambiando dinámicamente.

A diferencia del diseño de una aplicación web tradicional, React se basa en un paradigma denominado "programación orientada a componentes" siendo cada componente una parte visual de la aplicación web con la que el usuario puede interactuar.

Los componentes están escritos con la sintaxis JSX que permite escribir HTML y CSS dentro de funciones JavaScript, de esta forma se tiene un componente formado por un HTML con el estilo gráfico de CSS y toda la funcionalidad de JavaScript, que puede ser reutilizado en cualquier otra aplicación.

Cada componente tiene un estado que le permite a React saber si se ha modificado o no y en caso de que el estado cambie, React vuelve a renderizar ese componente. Gracias a esto, la aplicación únicamente actualiza aquellos componentes que se modifican.

En el Anexo N se muestran los archivos que forman el *front-end* de la aplicación

En el diagrama de la Figura 15 se muestran los componentes que forman la aplicación SPA desarrollada.

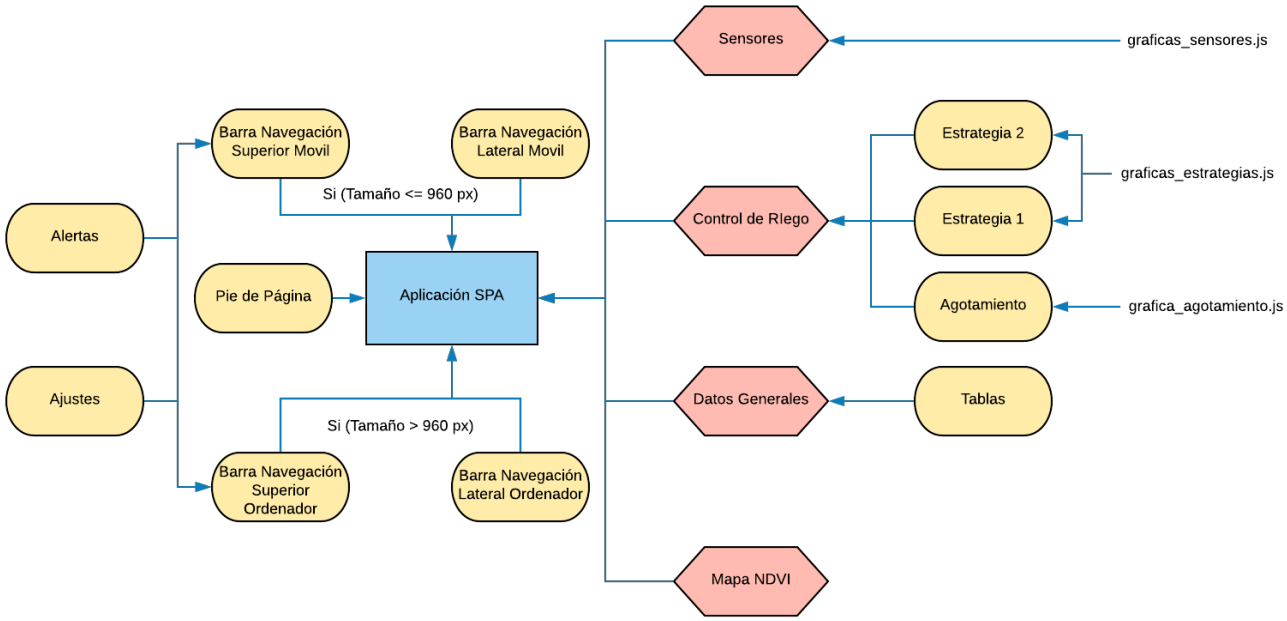

Figura 15 Diagrama de componentes de la aplicación web. Fuente: elaboración propia

Como se observa en el diagrama anterior, la aplicación web esta compuesta por varios componentes. Estos pueden representar una página de la aplicación web (componentes rojos) o una parte de esa página (componentes amarillos). De esta forma, como se observa en el diagrama, la aplicación web presenta una serie de componentes que son iguales en todas las páginas, como son: 1- Barra de navegación superior, 2- Barra de navegación lateral, 3- Pie de página. Sin embargo, los componentes representados en rojo van a visualizarse únicamente si se introduce una URL específica, dando la impresión de estar navegando por páginas diferentes. Además, los componentes pueden contener otros componentes, simplificando así el código y mejorando la escalabilidad de la aplicación.

3.2.3 Base de datos

Para que la aplicación web sea dinámica, es decir, se ajuste a cada usuario cada vez que este se conecta, necesita una base de datos para almacenar todo el contenido. En ella se guardan los valores de los sensores, variables del sistema de riego y de la explotación, datos de usuario y cualquier otro valor que le permita a la aplicación ajustar su contenido en cada momento.

Se han utilizado dos tipos de base de datos:

1. MySQL: Es una base de datos relacional, utiliza el lenguaje de consulta SQL (*Structured Query Language*).. Es la base de datos que se ha utilizado para casi toda la aplicación.

2. Redis: Es una base de datos de tipo no relacional, NoSQL, que almacena la información en forma de *clave-valor*. Se caracteriza por almacenar la información en la memoria caché, lo que permite acceder a ella de forma rápida. Se usa principalmente para almacenar datos que ocupan poco espacio y a los que es necesario acceder de forma rápida. Esta base de datos se utiliza para definir el periodo en el que se van a ejecutar tanto los scripts de programación del riego como los utilizados para descargar los valores de los sensores.

4. Implementación y despliegue de la aplicación web

En este capítulo se explica la metodología seguida, así como todas las herramientas que se han utilizado para la implementación de la aplicación web cuyo diseño se ha detallado en los capítulos anteriores.

4.1 Programación de riego

Como se vio en el diseño, para el cálculo de la programación de riego es necesario extraer los valores climatológicos de los próximos 7 días. Hay que conectarse por un lado a la AEMET, a través de su API, y por otro lado al sitio web de ElTiempo.es.

Para conectarse a la API de la AEMET se necesita una API key, que es una clave de autenticación de usuario para acceder al contenido de la API. Una vez que se dispone de la clave se accede a la API a través de una URL en la que se va a especificar los parámetros necesarios. Para automatizar el proceso, se descargan los valores a través de un script creado en Python. Se ha utilizado la librería Request. Esta librería permite realizar solicitudes HTTP (HTTP Request) a la API de AEMET, pasando como parámetro la clave mencionada anteriormente. En respuesta la AEMET envía a su vez dos URLs, una contiene los datos solicitados y otra con los metadatos, ambos en formato JSON. En el Anexo F se muestra un fragmento de código utilizado para conectarse a la API

Puesto que la AEMET no proporciona toda la información necesaria para realizar la programación del riego, los datos de precipitación y porcentaje de nubes se han extraído de ElTiempo.es. Para ello ha sido necesario conectarse a su sitio web y extraer dichos datos utilizando la técnica Web Scraping. Para ello, se ha utilizado, al igual que en el caso anterior, la librería *Request* para realizar la petición y la librería *BeautifulSoup*.

También es necesario obtener los valores medidos por los sensores de humedad y conductividad eléctrica que están instalados en la finca. Estos valores se encuentran almacenados en una base de datos propiedad de la empresa suministradora de dichos sensores. Únicamente se puede acceder a los datos mediante la web de la empresa, por lo que se ha desarrollado un script en Python que, mediante Web Scraping, se descarga los valores de los sensores y los introduce en la base de datos de la aplicación.

El código utilizado para descargar los valores de elTiempo.es es similar al usado para obtener los datos de los sensores. Sin embargo, este último se complica ya que es necesario acceder previamente con un usuario y una contraseña. Los formularios web tienen unos *tokens* de seguridad. El *token* es una clave aleatoria que solo el usuario y el servidor conoce y se utiliza para evitar ataques de terceros mediante CSRF (Cross-site request forgery). Para enviar el formulario con el usuario y contraseña es necesario rescatar primero el *token* y enviarlo al servidor junto con el usuario y la contraseña (Anexo D). Para ello, Se ha utilizado la librería *BeautifulSoup*, la librería estándar *urllib* para generar la petición y la librería *http.cookiejar* para gestionar las cookies. Estas últimas permiten mantener abierta la sesión del usuario para navegar por el sitio web.

Una vez obtenidos todos los valores, se envían a la base de datos MySQL. Para el envio de dichos valores se utiliza la biblioteca *mysql* de Python. Los parámetros que se necesitan para acceder a la base de datos y enviar los valores son:

- Nombre de la base de datos
- Usuario
- Contraseña
- Host
- Puerto
- Código SQL

En el Anexo G se expone un fragmento del código utilizado para conectarse con la base de datos

Finalmente, una vez obtenidos todos los valores se crearán un algoritmo con las ecuaciones expuestas en el apartado 3.1.1. Para ello se utilizará la librería estándar de Python *math* que ofrece todo tipo de funciones matemáticas.

4.2 Aplicación web

Para no extenderse demasiado, y hacer más sencilla su comprensión, se van a explicar los pasos seguidos para la implementación de la tabla *"NumeroDatos", desde* su creación en el *back-end* hasta su despliegue visual en el *front-end*, así como las interacciones entre ambos que permiten modificar sus valores. La implementación del resto de la aplicación se realiza de forma similar al ejemplo detallado con la tabla *"NumeroDatos"*.

4.2.1 Back-end

En el Anexo I se exponen los pasos iniciales para la creación del proyecto.

4.2.1.1 Implementación de la base de datos

Una vez creado el contenedor *Docker* donde se encuentra la base de datos (Anexo E), se configura el servidor web. Para ello se modifica el script donde se encuentran los ajustes de Django (*setting.py*) introduciendo la siguiente configuración:

Setting.py:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': MYSQL_NAME,
        'USER': MYSQL_USER,
        'PASSWORD': MYSQL_PASSWORD,
        'HOST': 'db', # database container name
        'PORT': '3306',
    }
}
```

Para conectarse con Python a la base de datos se necesitan dos librerías adiciones: *PyMySQL* y *mysql-connector-python.*

En el caso de la base de datos Redis solo hay que especificar la imagen de Docker e instalar el módulo *redis (Anexo E)*.

4.2.1.2 Implementación modelo

El modelo implementa las tablas de la base de datos y se definen en un script denominado *models.py*. En él se especifican las tablas que van a formar la base de datos. A continuación, se muestra el código para la creación de la tabla donde se almacenarán los ajustes de las gráficas.

Models.py:

```
class NumeroDatos(models.Model):
    Numero = models.IntegerField(blank=True, null=True)
    Todos = models.BooleanField(default=True)
    Conductividad = models.BooleanField(default=True)
```

Este código crea una tabla denominada "*NumeroDatos"* y contiene el campo *"Numero"* con formato entero, y los campos "*Todos*" y "*Conductividad*" en formato booleano. Es posible definir opciones como dar un valor por defecto, forzar que no se puedan tener valores nulos, permitir campos en blanco, etc.

Finalmente, la implementación de las tablas relacionadas con el control de usuarios se define en el Anexo J.

4.2.1.3 Implementación de la API REST

Definidas las tablas, es necesario desarrollar la API REST de la aplicación web para su conexión con todo la parte del front-end. Para implementar la API REST se va a utilizar la herramienta "Django Rest Framework (DRF)".

En primer lugar, se crea la configuración de la API en el archivo *setting.py* (Anexo K) de Django. Este archivo configura todo el comportamiento del proyecto, es donde se define la base de datos, las librerías externas utilizadas, los ajustes de seguridad, etc.. En segundo lugar, se definen los siguientes scripts:

- *serializers.py*: en el cual se definen los serializadores
- *views.py*: en el que se definen las vista
- *url.py*: en el que se definen los enrutadores

En el archivo *serializers.py* se crea un serializador para cada tabla de la base de datos. A continuación, se muestra un ejemplo del serializador usado para transformar los datos de la tabla "*UltimosValores*":

Serializers.py:

```
class NumeroDatosSerializador(serializers.ModelSerializer):
    class Meta:
        model = NumeroDatos
        fields = ('Numero', 'Todos', 'Conductividad')
```

Como ya se explicó en el apartado de diseño, estos serializadores también permiten modificar un valor antes de insertarlo. Para ello se usa el método "*to_internal_value*" que toma como parámetros los datos enviados por el *front-end* y devuelve un nuevo valor que se inserta en la base de datos. En el Anexo L, se muestra el método utilizado para modificar los valores de AFA, ADT y Kr. Para ello se han creado dos métodos adicionales, "*disponibilidad_agua"* y "*coeficiente_kc",* con las ecuaciones descritas en el apartado 3.1.2 del presente capítulo.

Una vez definidos los serializadores es necesario desarrollar la lógica de la API a través de las vistas. Para ello se utilizan dos clases definidas en DRF: una que únicamente mostrará en formato JSON los valores de la tabla, denominada *"ListAPIView"*, y otra que permite mostrar, insertar y actualizar los datos, denominada *"ModelViewSet"*. Ambas clases toman como parámetros el serializador definido anteriormente y una lista de objetos (QuerySet), que contiene los valores de la base de datos. Para las estrategias y los sensores se usará la primera, ya que solo se necesita obtener los valores, mientras que para las tablas que definen las características de la explotación, el sistema de riego y los ajustes de la aplicación se utilizará la segunda, permitiendo que el *front-end* pueda comunicarse con el *back-end* para actualizar la base de datos.

Continuando con el ejemplo anterior se muestra la vista creada para leer y modificar la tabla *"NumeroDatos"*. Además, se muestra un ejemplo de la vista utilizada para obtener los valores de la estrategia 1 del sector 1, en el que solo es necesario leer los valores.

Views.py:

```
class NumeroDatosViewSet(
        viewsets.ModelViewSet):
    queryset = NumeroDatos.objects.all()
    serializer_class = NumeroDatosSerializador


class Estrategia1Sector1View(ListAPIView):
    queryset = Estrategia1Sector1.objects.all()
    serializer_class = Estrategia1Sector1Serializer
```

Finalmente hay que definir la URL de cada una de las APIS definidas en las vistas utilizando los enrutadores. Estos se definen en el archivo urls.py. A continuación, se muestra un ejemplo del enrutador utilizado para definir la URL de la vista encargada de la tabla "*NumeroDatos*"

Urls.py:

```
router = routers.DefaultRouter()
router.register('ajustes/numero_datos', NumeroDatosViewSet)


urlpatterns = [
    path('', include(router.urls)),
]
```

4.2.1.4 Implementación de la programación de riego

Por último, hay que implementar en la aplicación web el algoritmo para calcular la programación del riego. Este algoritmo está compuesto por cinco módulos:

1. Obtención de datos de la AEMET
2. Obtención de datos de elTiempo.es
3. Cálculo del agotamiento de humedad en el suelo
4. Cálculo de las estrategias de riego
5. Descarga de los valores subidos calculados por los sensores de humedad

La ejecución de estos cinco módulos debe realizarse de forma periódica. Así, los cuatro primeros módulos se ejecutan de forma diaria, a las 6:30 de la mañana, mientras que el quinto módulo es ejecutado cada 15 min (periodo de espera del sensor entre envío de datos).

Para poder llevar a cabo esta periodicidad de ejecución de funciones, se ha utilizado la librería *Celery* de Django que permite programar tareas. Esta librería se comunica con la base de datos Redis, para establecer el periodo de ejecución de dichos módulos.

Para poder modificar de forma fácil y dinámica el tiempo y la frecuencia de ejecución de esto módulos, se ha instalado *django-celery-beat* que permite gestionar Celery usando la interfaz de administración de la aplicación web. En el Anexo M se muestra los pasos seguidos para configurar Celery.

4.2.2 Front-end

Una vez definida la lógica de la aplicación en el *back-end*, hay que diseñar la interfaz de usuario utilizando la librería React. Todos los componentes de React se implementan en una función denominada *"render"* que toma como parámetros los componentes de la aplicación, definidos en el archivo *App.js* y el lugar de la plantilla de Django (HTML), donde se va a desplegar React.

A continuación, se muestra dicho componente, definido en el archivo *index.js*.

Index.js

```
ReactDOM.render(
    <React.StrictMode>
        <BrowserRouter>
            <App />
        </BrowserRouter>
    </React.StrictMode>

    , document.getElementById('app')
);
```

Todos los componentes de la interfaz están en el archivo *App.js* y se definen mediante funciones que permiten desplegar componentes sin crear clases. Actualmente es el método más utilizado, ya que simplifica la sintaxis.

Para acceder a las funcionalidades que ofrece React es necesario recurrir a los *hooks*. Los *hooks* son unas APIs de la librería React que permiten acceder al estado y otras características de la aplicación. Se han utilizado dos:

- useState: almacena el estado del componente
- useEffect: acepta una función como argumento que se ejecuta cuando el componente se renderiza por primera vez y cada vez que dicho componente se actualice, es decir, cambie de estado.

En el Anexo O se muestra un ejemplo del diseño del componente encargado de recibir los ajustes de las gráficas

Una vez definidos los componentes es necesario crear las rutas de nuestro *front-end*. Para ello se ha usado la librería *ReactRouter* que permite renderizar un componente u otro en función de la URL que visitemos, de modo que, aunque la aplicación realmente está formada por una sola página, se tiene la sensación de estar navegando por distintas.

Las gráficas se han desplegado utilizando la librería *Highcharts*, que permite crear gráficas interactivas en JavaScript.

Para mejorar el diseño de la interfaz se ha utilizado la librería *material-ui* que se fundamenta en la normativa Material-Design (https://material.io/design), desarrollada por Google.

Una vez terminado el desarrollo de la aplicación con React se procede a diseñar la parte de registro e identificación de usuarios. Para conseguir una interfaz amigable, limpia y perfectamente adaptable a todo tipo de dispositivos se han utilizado una serie de herramientas que se mostrarán a continuación:

1. *Bootstrap*: es un *framework* CSS3 que permite crear interfaces web adaptativas. Es decir, permite que la aplicación web se adapte perfectamente a diferentes tamaños de pantalla

2. *Django-crispy-forms*: es una librería que ayuda en la gestión de formularios en las plantillas HTML.

3. *Jquery*: es una librería de JavaScript que facilita el manejo de eventos, la interacción con el DOM (Document Object Model) y la creación de llamadas AJAX (Asynchronous JavaScript And XML).

4.3 Despliegue

Finalizado el diseño de la aplicación, es necesario pasar a la fase de producción. Esta fase consiste en desplegar la web en un servidor permitiendo que esté operativa las 24 horas y que pueda ser accesible desde cualquier parte. En este apartado del proyecto hay que configurar el servidor web e implementar las medidas de seguridad.

Se ha utilizado una Raspberry pi 3+ como servidor. La aplicación se ha desarrollado para ser lo más eficiente posible, permitiendo su funcionamiento en este "mini-ordenador". La elección de este servidor, atiende a aspectos económicos..

En la Figura 16 se muestra un esquema de la arquitectura que va a seguir la aplicación web.

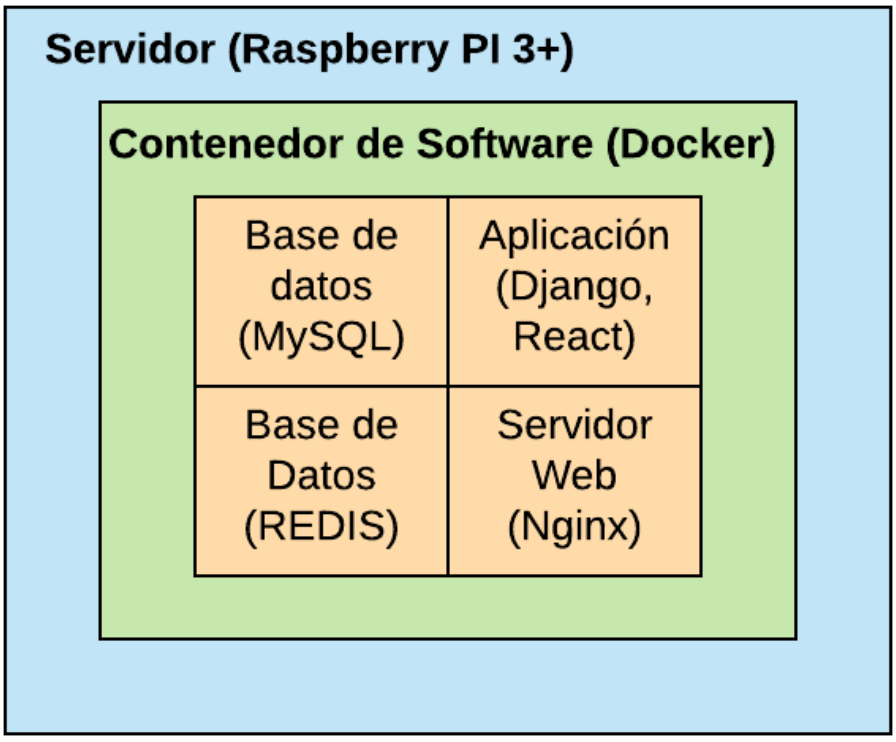

Figura 16 Arquitectura de la aplicación web en producción. Fuente: elaboración propia

4.3.1 Servidor web

Para la configuración del servidor web en producción se ha utilizado Gunicorn y Nginx.

Gunicorn es un servidor WSGI (Web Server Gateway Interface) de aplicaciones. Este permite traducir las solicitudes HTTP de los usuarios, a llamadas Python que la aplicación puede procesar. Para configurarlo únicamente hay que instalarlo e iniciar el servidor con el comando:

```
gunicorn dashboard_project.wsgi:application --bind 0.0.0.0:8000"
```

Nginx es un servidor que realiza las funciones de proxi inverso, permitiendo mejorar la seguridad y el rendimiento. El código de configuración de Nginx se muestra en el Anexo P.

4.3.2 Seguridad de la aplicación (Django)

Uno de los aspectos más importantes al pasar una aplicación web a producción es la seguridad. Las aplicaciones web son totalmente accesibles, lo que supone un blanco potencial para los ciberataques. Además, existen una gran cantidad de *softwares* automatizados (bots) que intentan buscar vulnerabilidades para atacar el servidor.

Aunque no se va a entrar en profundidad en las medidas de seguridad que se han aplicado, ya que queda fuera del alcance de este proyecto, se van a nombrar las principales:

- Certificado SSL: este permite la transmisión de información encriptada entre el usuario y el servidor web. Se ha instalado un certificado perteneciente a la autoridad de certificación Let's Encrypt ya que proporciona certificados gratuitos. Para instalar dicho certificado habrá que modificar la configuración del servidor Nginx (Anexo P).

- Host permitidos: solo se podrá acceder únicamente a la aplicación desde el dominio correspondiente.

- Protección contra XSS (Cross Site Scripting o falsificación de petición en sitios cruzados): evita que se puede almacenar en la base de datos pequeños trozos de código.

- Protección contra CSRF (Cross-Site Request Forgery o falsificación de petición en sitios cruzados): evita que un atacante provoque que un usuario registrado ejecute una acción no deseada.

- Protección contra clickjacking (secuestro de clic): evita que el usuario pueda dar click en enlaces no seguros

- Cookies seguras: obliga a que las cookies se envían utilizando una conexión segura SSL.

- Protección de la política de referencia (Referrer Policy): controla la información que se envía en la cabecera HTTP cuando se visita otro sitio web a través de un enlace. De esta forma se evita que se envíe información sensible, como la dirección del servidor.

4.3.3 Servidor (Raspberry Pi)

Finalmente habrá que instalar todo el proyecto en la Raspberry PI. Como ya se ha comentado, el uso de Docker permite que podamos desplegar la aplicación en cualquier sistema operativo que tenga instalado dicho software.

La imagen MySQL utilizada para la creación de la base de datos no está disponible para procesadores ARM, que es el que tiene instalado la Raspberry PI. Por ello se ha utilizado una imagen que instala la base de datos MariaDB, que es un sistema de base de datos derivado de MySQL. Sin embargo, puesto que sigue la misma sintaxis, no habrá que realizar más modificaciones.

Capítulo 4 - Aplicación de la herramienta en caso real

En este capítulo se exponen las diferentes pantallas de la aplicación web en modo producción.

1. Módulo de gestión de usuarios

En el módulo de gestión de usuarios (Figura 17) se tienen las funciones de inicio de sesión y cambio de contraseña de los usuarios.

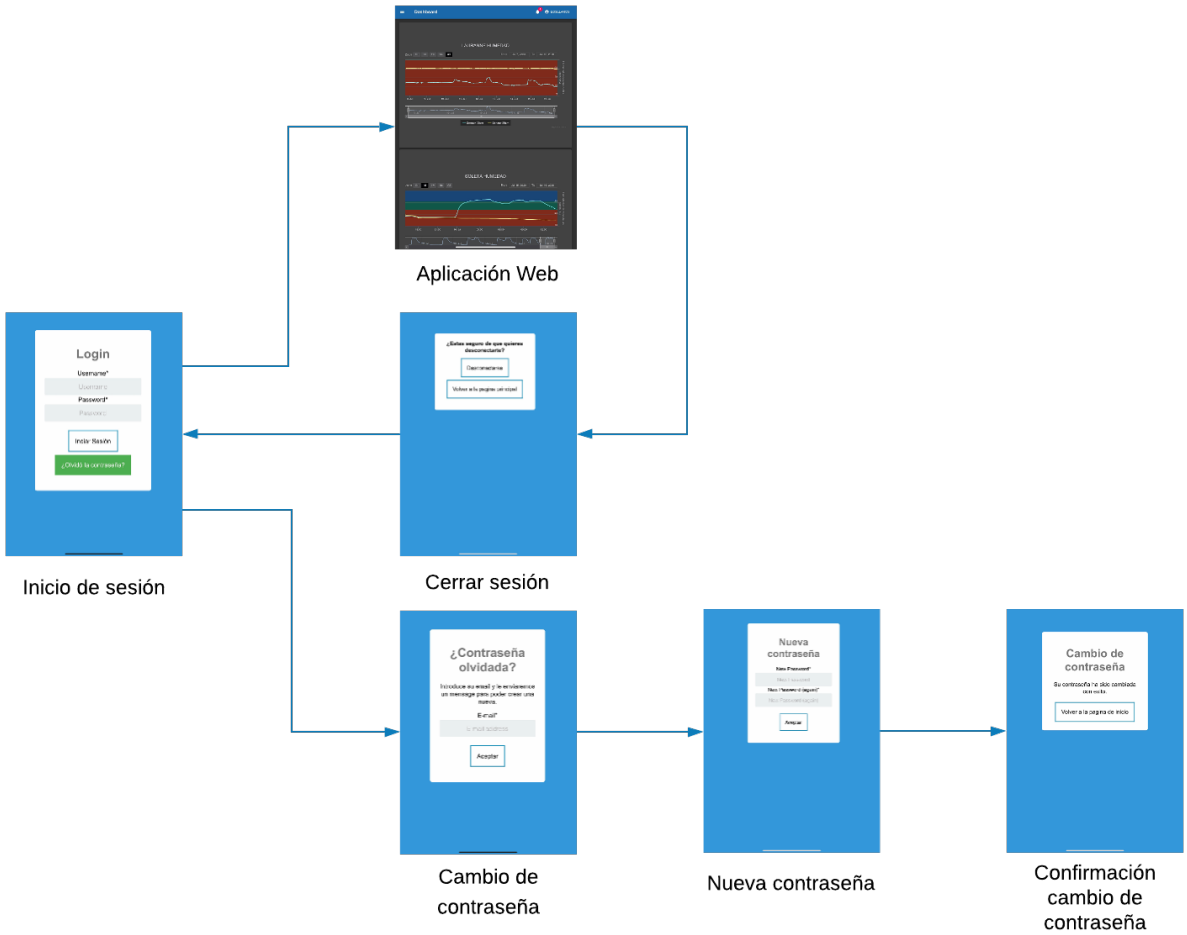

Figura 17 Navegación inicio de sesión. Fuente: elaboración propia

En el panel de inicio de sesión el usuario tendrá que rellenar los campos usuario y contraseña, procediendo después a su autentificación por parte de la aplicación.

En el panel de cambio de contraseña el usuario debe introducir el email que tiene asociado a su cuenta. Se le enviara un correo con una URL que le redirigirá a la ventana de nueva contraseña, donde se le pedirá que introduzca dos veces la nueva contraseña.

El registro de usuarios no se ha implementado ya que, por cuestiones de seguridad, solo podrá registrar usuarios el administrador de la aplicación. Para ello utilizará el panel de administración de Django (Figura 18).

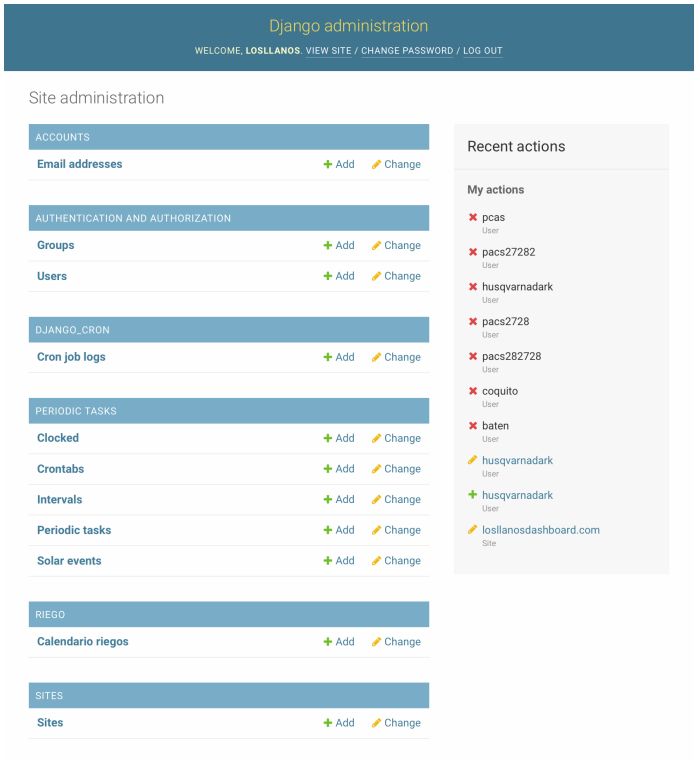

Figura 18 Panel de administración Django. Fuente: elaboración propia

El panel de administración permitirá, entre otras cosas, al administrador de la aplicación:

- Crear usuarios
- Eliminar usuarios
- Administrar la base de datos
- Cambiar el tiempo de descarga de datos de los sensores
- Cambiar el número de veces que se calculan las distintas estrategias de riego

2. Módulo Aplicación Web SPA

Este módulo está formado por toda la aplicación web que se ha desarrollado en React.

El diseño de está interfaz cambia en función del tamaño de pantalla que tenga el dispositivo utilizado. En pantallas grandes, de más de 960 px (pixeles), la barra de navegación superior y lateral siempre se encuentra visible (Figura 19), pudiendo esta ultima esconderse en caso de querer visualizar las gráficas a más tamaño. Cuando el dispositivo presenta una pantalla de menos de 960 px la barra de navegación lateral se esconde y aparece un botón en la barra de navegación superior que permite desplegarla (Figura 20), mejorando así la visualización el contenido. Además, para mejorar aún más la visibilidad en este tipo de pantallas de menor tamaño, la barra de navegación superior se esconde cuando se navega en dirección descendente, apareciendo nuevamente cuando se desliza hacia arriba.

Figura 19 Diseño para pantallas de más de 960 px. Fuente: elaboración propia

Figura 20 Diseño para pantallas de menos de 960px. Fuente: elaboración propia

La aplicación está formada por diferentes pantallas que muestran información referente al riego, los sensores y la finca. Estas pantallas comparten componentes como las barras de navegación y el pie de página. En la Figura 21 se muestra la estructura que sigue la aplicación

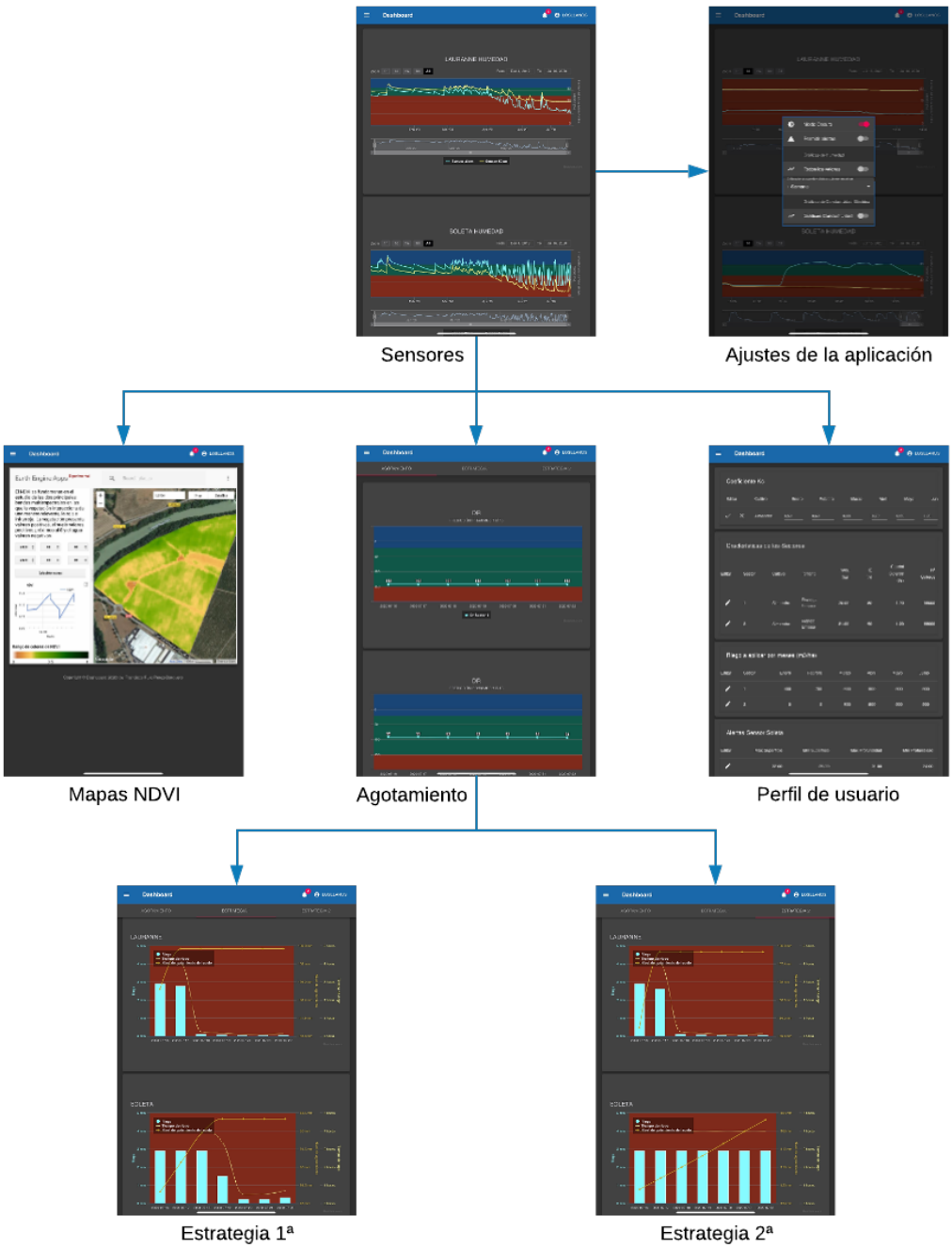

Figura 21 Estructura de la interfaz. Fuente: elaboración propia

Las pantallas que se muestran en la figura anterior realizan las siguientes funciones que se mostrarán a continuación:

1. Sensores: es la pantalla principal de la aplicación web. Muestra los datos de humedad y conductividad eléctrica de los sensores instalados en la explotación (Figura 20)

2. Ajustes de la aplicación: muestra y permite cambiar los ajustes de la aplicación (Figura 22): Modo oscuro (activa o desactiva el modo oscuro), Alertas (activa o desactiva las alertas), Todos los valores (despliega todos los valores de los sensores de la base de datos) Número de datos (define el número de datos que se despliega en las gráficas si se ha desactivado la opción de "todos los valores". Permite elegir entre una semana, un mes y un año), gráficas de conductividad (activa o desactiva las gráficas de conductividad eléctrica. Si están desactivadas, el número de datos que la aplicación tiene que descargar es menor, por lo que se reduce el tiempo de carga) y RDC (si se activa se calcula la estrategia de riego deficitario controlado)

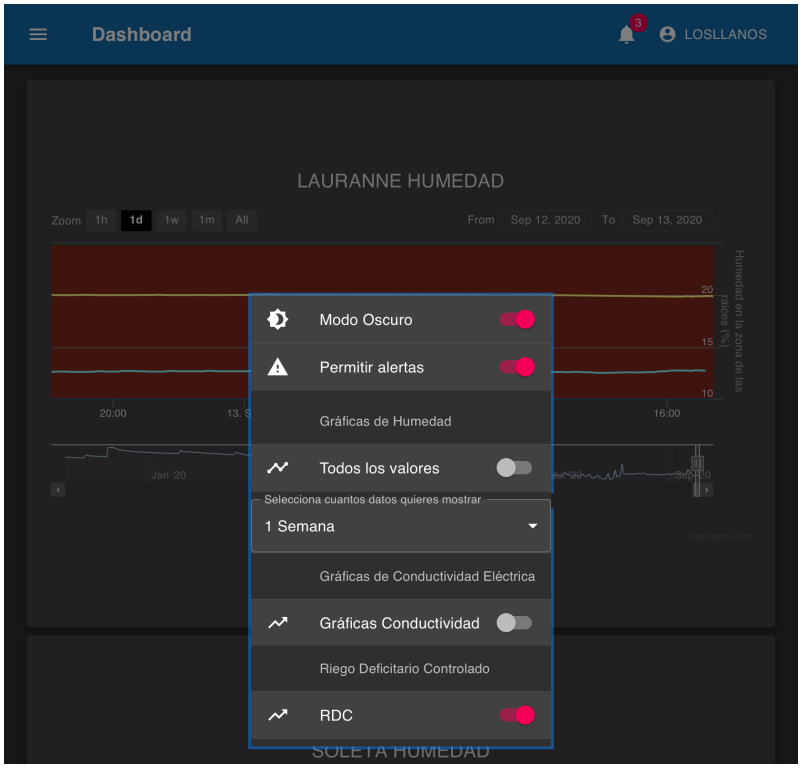

Figura 22. Pantalla de ajustes de la aplicación. Fuente: elaboración propia

3. Control de riego: muestra las pantallas que se encargan de calcular el riego a aplicar

- Agotamiento: muestra la gráfica de agotamiento de humedad en la zona radicular, en mm, para los próximos 7 días (Figura 23).

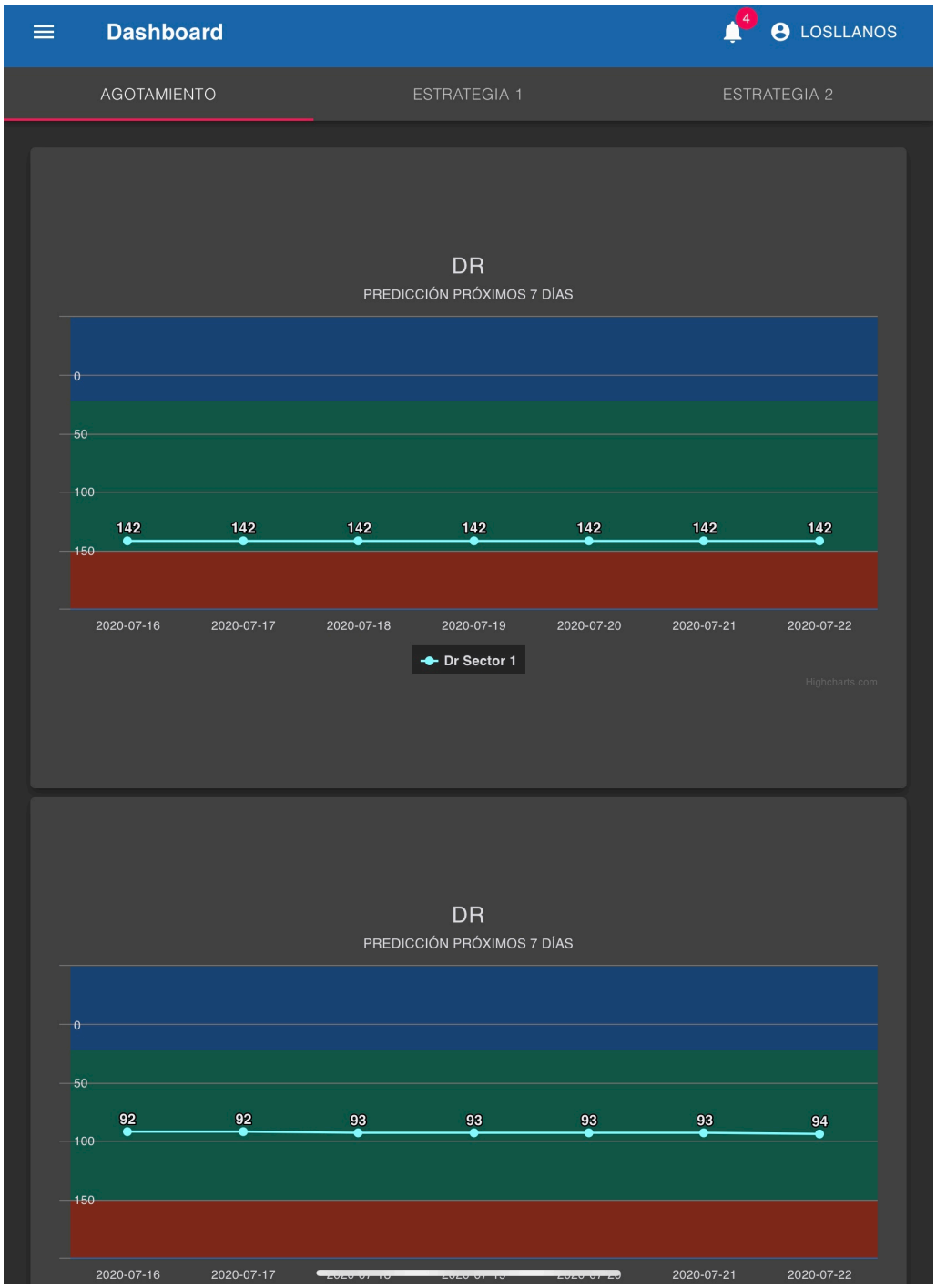

Figura 23 . Pantalla de agotamiento de humedad. Fuente: elaboración propia

- Estrategia 1ª: Muestra los datos, en gráficas y tablas, utilizando el valor de Dr obtenido al realizar el balance de agua del día anterior (Figura 24). En las gráficas se muestrann los milímetros que hay que aplicar de riego, el tiempo que el sistema de riego debe estar funcionando para aplicar dicha agua y el agotamiento del suelo del día siguiente si se siguiera dicha estrategia.

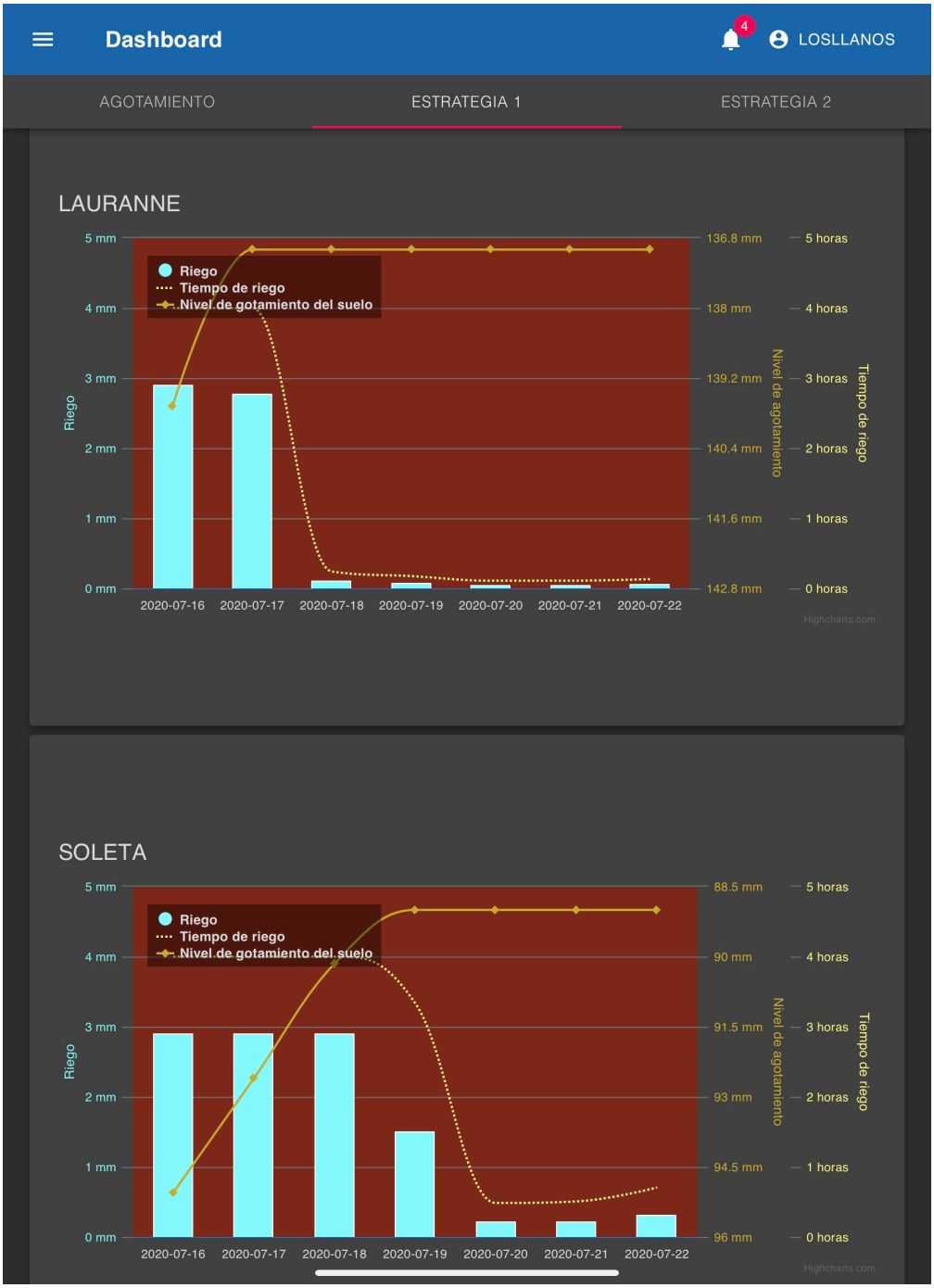

Figura 24 Pantalla de estrategia 1. Fuente: elaboración propia

- Estrategia 2ª: muestra los datos, en gráficas y tablas, utilizando el valor que dan los sensores para saber el agotamiento de humedad en el suelo (Figura 25). Como se observa en la siguiente figura, la cantidad de agua que hay que aplicar durante la semana en el sector Soleta es mayor que en la estrategia primera (Figura 26). Esto es debido a que el valor de agotamiento que muestra el sensor es mayor que el que se predijo para ese día, ya que no se regó el día anterior. Esta estrategia muestra un escenario más actual, pero a la vez más dependiente del valor que los sensores aportan.

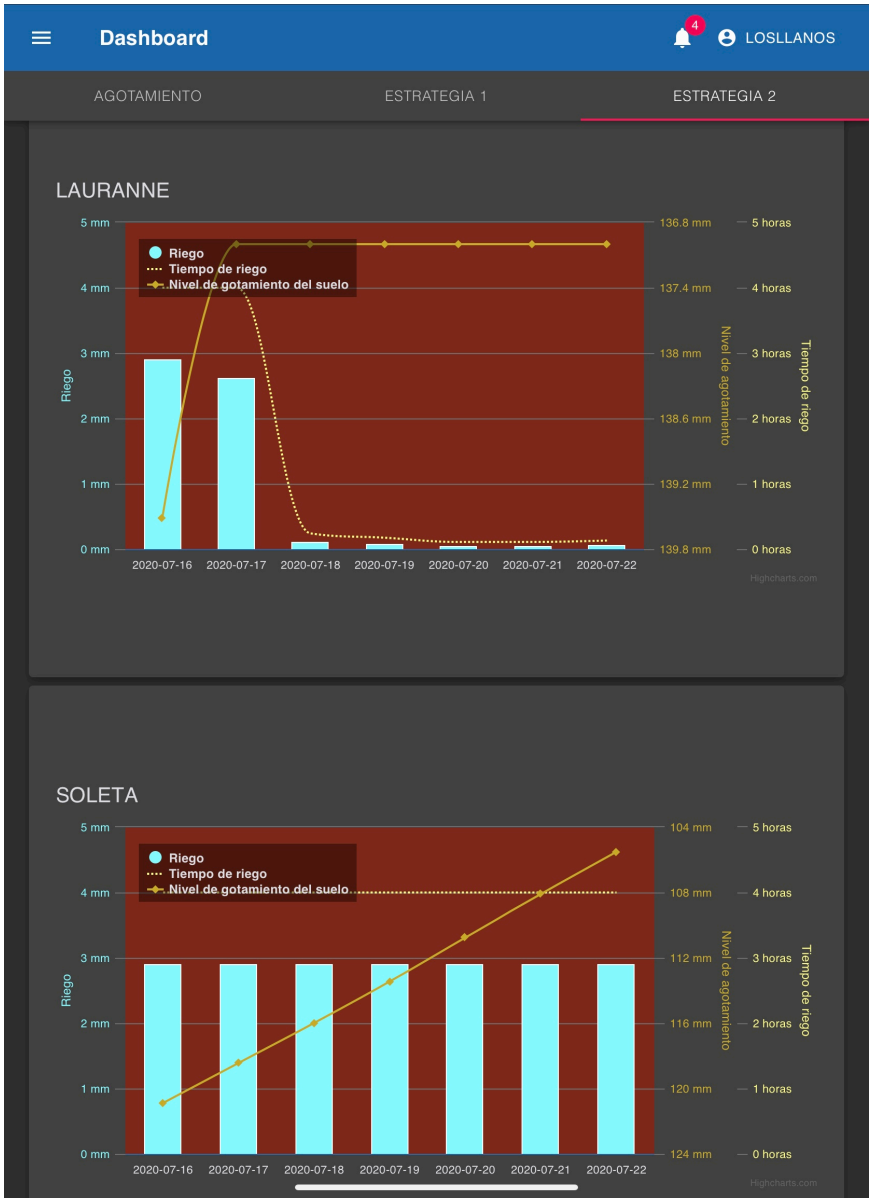

Figura 25 Pantalla de estrategia 2. Fuente: elaboración propia

- Perfil de usuario: permite visualizar y cambiar todos los valores referentes al cultivo y a la explotación. Estos valores son necesarios en el cálculo del agotamiento del suelo y las diferentes estrategias y son:

  - Tabla característica sector: permite visualizar y cambiar las siguientes características del sector: Cultivo, terreno, área, eficiencia del riego, caudal de los goteros, nº de goteros, profundidad de las raíces, diámetro de copa, arboles por Ha, tiempo de riego máximo, Dr objetivo de riego, ADT y AFA

  - Dr objetivo por meses: Establece el agotamiento máximo de humedad por meses. Esta tabla es utilizada cuando la opción de RDC esta activada

  - Tabla coeficiente Kc: permite cambiar el coeficiente de cultivo

  - Riego requerido: permite visualizar y cambiar el riego que se quiere aplicar cada mes. Esto es similar al porcentaje de necesidades de riego ya que también limita el agua máxima que se puede aplicar.

  - Alertas sensor: permite definir las alertas de humedad de los sensores.

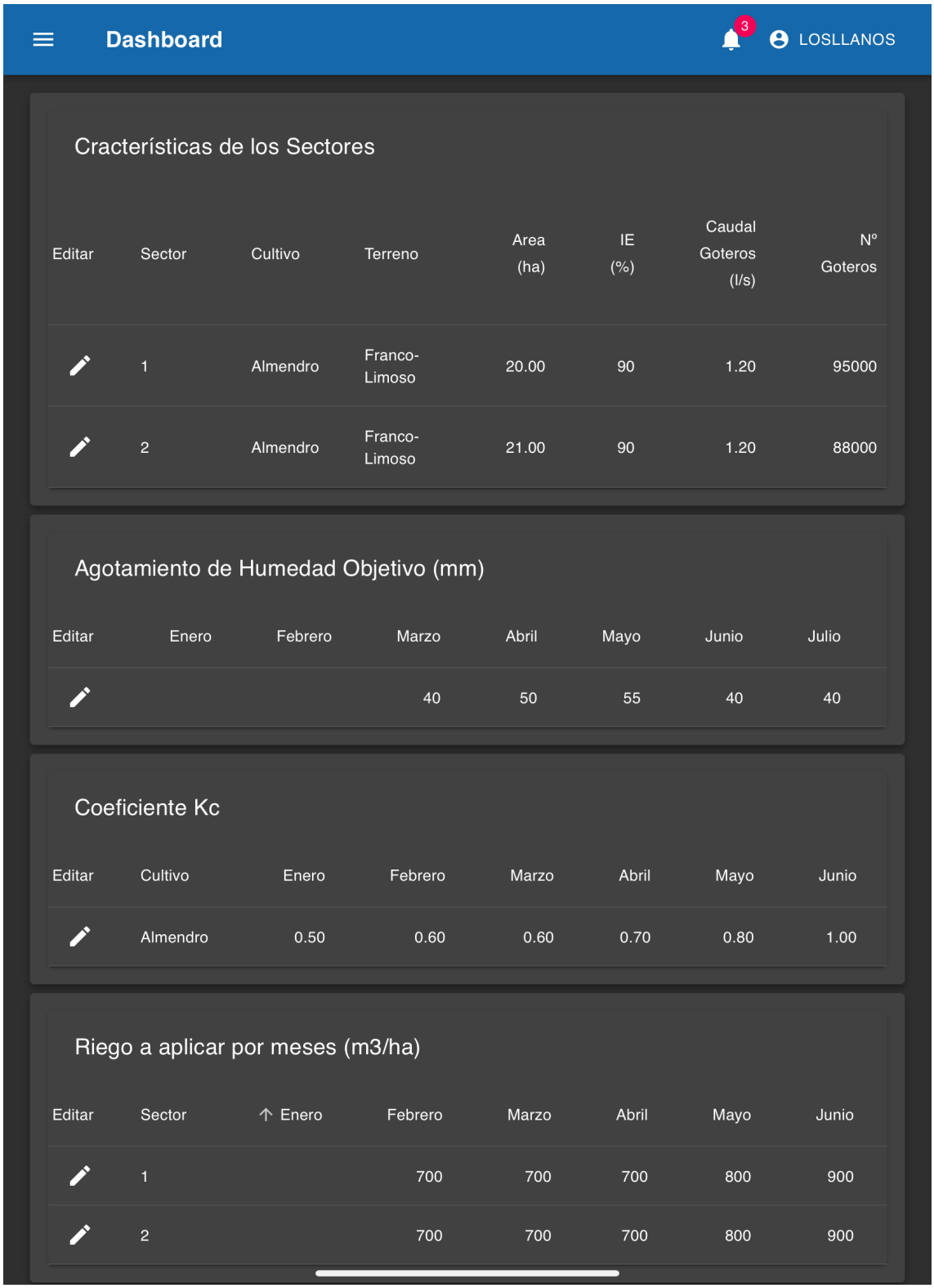

Figura 26 Pantalla perfil de usuario 2. Fuente: elaboración propia

- Mapa NDVI: Utiliza la aplicación web Google Earth Engine en la que se ha programado un script en JavaScript que despliega un mapa NDVI de las fechas seleccionadas(Figura 27) (Anexo Q)

Figura 27 Pantalla de Mapa NDVI 2. Fuente: elaboración propia

Los componentes comunes en todas las pantallas son:

- Barra de navegación superior: permite acceder a la barra de navegación lateral, visualizar las alertas, acceder al panel de administración, acceder a los ajustes y desconectarse (Figura 28)

Figura 28 Barra navegación superior móviles. Fuente: elaboración propia

- Barra de navegación lateral: permite navegar entre las diferentes páginas de la web (Figura 29)

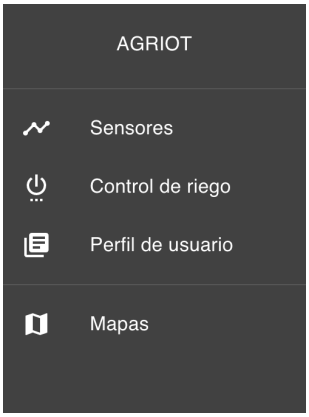

Figura 29 Barra navegación lateral móviles. Fuente: elaboración propia

- Pie de página: muestra información adicional de la aplicación

Capítulo 5 - Conclusiones

- En este trabajo final de máster se ha desarrollado una aplicación web para controlar en tiempo real un riego de precisión, permitiendo una programación óptima, sencilla y personalizada para diferentes estrategias.

- Se ha automatizado el cálculo de las necesidades de riego según la metodología de FAO. haciendo uso de las APIS de la AEMET y elTiempo.es, la aplicación realiza una predicción semanal de la variación de la humedad en el suelo.

- La aplicación web implementa también el cálculo del tiempo que el sistema de riego debe funcionar a diario para aplicar el riego de precisión.

- La aplicación ha sido desarrollada haciendo uso de dos de las herramientas más utilizadas hoy en día en desarrollo web (ReactJS y Django). Esto permite que dicho proyecto sea ampliamente escalable y útil de cara al mundo profesional.

- Finalmente, la aplicación se ha pasado a modo producción y se ha instalado en un miniordenador, permitiendo así la instalación rápida, económica y segura en la explotación.

Capítulo 6 - Líneas de futuro

A continuación, se muestran las principales líneas de investigación que podrían desarrollarse a partir de los resultados obtenidos en el presente trabajo.

- Seguimiento de las diferentes estrategias calculadas por el sistema a lo largo de un ciclo hídrico completo, comparando la variación de humedad predicha en el suelo, con la real.

- Incorporación de otro tipo de sensores, como una estación meteorológica, que mejoren la precisión de la programación del riego

- Incorporación a la aplicación de las principales redes nacionales o autonómicas para la detección de plagas y enfermedades

- Estudio e implementación de tecnologías relacionadas con la inteligencia artificial que permita ajustar la variación de humedad en el suelo, comparando lo calculado y lo real y utilizando las variables climáticas para ajustar dicho valor.

- Despliegue de la aplicación web en servidores en la nube de modo que haya la posibilidad de disponer de ella sin necesidad de tener instalado un mini-ordenador en la explotación.

Capítulo 7 - Bibliografía

1. Artículos

Alcaide Zaragoza, Carmen & García, I. & González Perea, Rafael & Camacho, Emilio & Rodríguez Díaz, Juan. (2019). REUTIVAR: Model for Precision Fertigation Scheduling for Olive Orchards Using Reclaimed Water. 11. 1-17. 10.3390/w11122632.

Fernández, J.E.. (2014). Plant-based sensing to monitor water stress: Applicability to commercial orchards. Agricultural Water Management. 142. 99–109. 10.1016/j.agwat.2014.04.017.

Fernández García I., Gonzalez Perea R., Martín Arroyo M., Rodríguez Díaz J.A., Camacho E., Montesinos P. (2016). Programación de un riego de precisión mediante el uso de las TICs. Aplicación en el cultivo de la fresa. http://www2.ual.es/SNIH16/web/Web/5-17.pdf

Steduto, Pasquale & Hsiao, Theodore & Fereres, E. & Raes, Dirk. (2012). Crop yield response to water. Food Agric. Org. United Nations. 6-9.

Arroyo, M. & García, I. & González Perea, Rafael & Morillo, Jorge & Rodríguez Díaz, Juan & Camacho, Emilio & Montesinos, P.. (2015). EL RIEGO DE PRECISIÓN EN EL CULTIVO DE FRESA EN LA PROVINCIA DE HUELVA. 363-372. 10.4995/CNRiegos.2015.1462.

Allen, Richard & Pereira, L. & Raes, D. & Smith, M.. (2006). Evapotranspiración del cultivo. Guías para la determinación de los requerimientos de agua de los cultivos. Estudio FAO Riego Y Drenaje No 56.

Fereres, E., Soriano, M.A., (2007). Deficit irrigation for reducing agricultural water use. Journal of Experimental Botany, Volume 58, Issue 2, 1 January 2007, Pages 147–159

García Morillo, J. & Rodríguez Díaz, Juan & Montesinos, Pilar (2015) [Tesis doctoral]. Hacia el riego de precisión en el cultivo de fresa en el entorno de Doñana

Allen, R.G., Pereira, L.S., Raes, D., Smith, M. (1998) Crop evapotranspiration —guidelines for computing crop water requirements. FAO Irrigation and drainage paper 56. Food and Agriculture Organization, Rome.

2. Páginas Web

INSTITUTO NACIONAL DE ESTADÍSTICAS

https://www.ine.es/dyngs/INEbase/es/categoria.htm?c=Estadistica_P&cid=1254735976602

FAO: Textura del suelo

http://www.fao.org/tempref/FI/CDrom/FAO_Training/FAO_Training/General/x6706s/x6706s06.htm

JUNTA DE ANDALUCIA: Fundamentos del riego (2015)

https://www.juntadeandalucia.es/export/drupaljda/1337160941Fundamento_del_riego_1.pdf

IRNAS: Estrategias y programación del riego

http://digital.csic.es/bitstream/10261/130206/1/Manual%20de%20riego_Jefer2015.pdf

GOOGLE EARTH ENGINE

https://earthengine.google.com/

REACTJS: Documentación oficial

https://reactjs.org/docs/hello-world.html

DOCKER

https://www.docker.com/

NGINX

https://www.nginx.com/

AEMET OPEN-DATA:

https://opendata.aemet.es/centrodedescargas/inicio

ELTIEMPO.ES: Córdoba

https://www.eltiempo.es/cordoba.html

3. Libros

William S. Vincent (2018). Django for Beginners: Learn web development with Django 2.0

William S. Vincent (2019). Django for Professionals: Production websites with Python & Django

Octavio Arquero (2013). Manual del Almendro. Consejería de Agricultura, Pesca y Desarrollo Rural

Anexos

Anexo A Herramientas de software utilizadas

SOFTWARE	VERSIÓN
macOS Catalina	**10.15.4**
Raspbian	**4.19**
Docker Desktop	**2.2.0.5**
Visual Studio Code	**1.47.1**

Anexo B Paquetes de Python instalados

PAQUETE	VERSIÓN
Django	**3.0.2**
Djanngo-allauth	**0.41.0**
Django-cryptographic-fields	**0.5.1**
Django-widget-tweaks	**1.4.5**
cryptography	**2.8**
django-crispy-forms	**1.9.0**
django-debug-toolbar	**2.2**
PyMySQL	**0.9.3**
mysql-connector-python	**8.0.19**
requests	**2.23.0**
beautifulsoup4	**4.8.2**
celery	**4.4.2**
redis	**3.4.1**
django-celery-beat	**2.0.0**
django-cron	**0.5.1**
whitenoise	**5.0.1**
gunicorn	**20.0.4**

Anexo C Módulos de Node.js instalados

PAQUETE	VERSIÓN
react	**16.13.1**
React-dom	**16.13.1**
webpack	**4.43.0**
webpack-cli	**3.3.11**
@babel/core	**7.10.1**
@babel/plugin-proposal-class-properties	**7.10.1**
@babel/preset-env	**7.10.2**
@babel/preset-react	**7.10.1**
babel-loader	**8.1.0**
@material-ui/core	**4.10.0**
@material-ui/icons	**4.9.1**
@material-ui/styles	**4.10.0**
axios	**0.19.2**
highcharts	**8.1.0**
material-table	**1.60.0**
notistack	**0.9.17**
react-alert	**7.0.1**
react-alert-template-basic	**1.0.0**
react-headroom	**3.0.0**
react-router-dom	**5.2.0**

Anexo D Descarga de los valores de los sensores

Para descargar los valores de los sensores instalados en la finca hay que acceder al sitio web de la empresa., introduciendo el usuario y la contraseña, redirigirse a la página donde se encuentran las gráficas y descargar los últimos valores utilizando la técnica de *Web Scraping*.

Lo primero es obtener el Token de seguridad. Para ello se busca en el código HTML de la página de inicio de sesión. Una vez rescatado se realiza un método POST que contiene el usuario, la contraseña y el Token, realizando así el inicio de sesión en la página de los sensores. Es importante manejar las cookies para poder navegar por el sitio web una vez se ha iniciado sesión. Todo esto se consigue utilizando las librerías urllib y http.cookiejar.

```
def site_login(self):
        '''
        Este modulo rescata el Token de la web y hace un post
        con el usuario y la contraseña
        '''
        try:
            jar = http.cookiejar.FileCookieJar("cookies")
            self.opener = build_opener(HTTPCookieProcessor(jar))

            r = self.opener.open(self.url_login).read()
            soup = bs4.BeautifulSoup(r, "html.parser")

            hidden_tags = soup.find_all("input", type="hidden")
            # Obtiene el token de seguridad - que es el 5° campo
           oculto
            token_name = hidden_tags[1]['name']
            token_val = hidden_tags[1]['value']
            print('Token name = ', token_name)
            print('Token value = ', token_val)

            login_data = {'username': self.username,
                          'password': self.password,
                          'remember': 'yes',
                          token_name: token_val}

            values = urlencode(login_data)
            values_bin = values.encode('utf-8')
            # Inicio de sesión
            self.opener.open(
                self.url_post, values_bin)
            print(Inicio de sesión realizado')

        except Exception as ex:
            print('No se ha podido iniciar sesión: ', ex)
```

Realizado con éxito el inicio de sesión, se redirige a la página donde se encuentran las gráficas y se descargan los valores utilizando el siguiente código:

```
html_charts =
'window.plotalot_chart_{}_data.addRows\(\n\[(.*?)\]\);$'.format(
                id_grafica)
pattern = re.compile(
r"" + html_charts, re.MULTILINE | re.DOTALL)

# Se descargan todo los valores de las graficas en formato
string
script = self.html_final.find("script", text=pattern)
chart_data_text = pattern.search(script.text).group(1)

# Se separan los valores en cada '],'
array_all_data = chart_data_text.split('],')
self.array_datos = []

for i in array_all_data:
    x1 = i.replace('[', '').replace(
        ']', '').replace('new Date(', '')
    x2 = x1.split(')')
    # se hace un array separandolo por ','
    fecha = x2[0].split(',')
    # se eliminan los espacios en blanco de la derecha
    array_fecha = [i.replace(' ', '') for i in fecha]
    # Calculo de los milisegundos
    year = array_fecha[0]
    # Hay que sumarle 1 al mes, ya que el javascript le resta
     uno
    month = str(int(array_fecha[1]) + 1)
    day = array_fecha[2]
    hour = array_fecha[3]
    minute = array_fecha[4]
    # Se usa el modulo datetime para pasar a milisegundos
    fecha_str = "{},{},{},{},{}".format(
        day, month, year, hour, minute)

    date_obj = datetime.strptime(fecha_str,
                                 '%d,%m,%Y,%H,%M')
    # timestamp añade una hora mas en las graficas
    date_obj = date_obj - timedelta(hours=1)
    millisec = int(date_obj.timestamp() * 1000)

    # se hace una cadena separando por la ','
    values_Sensor = x2[1].split(',')
    # se elimina el primer elemento de la cadena
    values_Sensor.pop(0)
    # se cambia null por None
    array_values_sensor = [
        i if i != 'null' else None for i in values_Sensor]
```

```
array_final = [millisec,
               array_values_sensor[0],
                array_values_sensor[1]]
self.array_datos.append(array_final)
```

Descargados ya todos los valores de los sensores el script se conecta con la base de datos e inserta los valores que esta no contenga

Anexo E Instalación de los contenedores con Docker

En primer lugar, hay que definir los diferentes contenedores que va a tener la aplicación web. Es necesario instalar la ultima versión de Docker disponible en su página oficial.

A continuación, se crean cuatro archivos:

- Dockerfile: contiene todos los comandos que son necesarios para ensamblar una imagen
- docker-compose.yml: es una herramienta para definir e iniciar varios contenedores de forma automática
- requirements.txt: contiene todas las librerías con la versión correspondiente mencionadas en el Anexo B
- .env: contendrá las variables de entorno de la aplicación, como contraseñas, características de la base de datos, claves de seguridad, etc.

El archivo Dockerfile es el siguiente:

```
# Imagen de Python 3. Sistema operativo Linux
FROM python:3

# Aseguro que la salida de la consola se vea familiar
ENV PYTHONUNBUFFERED 1
# Python no escribirá archivos .pyc
ENV PYTHONDONTWRITEBYTECODE 1

# Se crea un directorio dentro de la imagen
RUN mkdir /code
# se define el directorio creado
WORKDIR /code

# se copia el archivo requirement en directorio code
COPY requirements.txt /code/
# se instalan las librerias
RUN pip install -r requirements.txt

# se copia el proyecto y se pega en code
COPY . /code/
```

El archivo docker-compose.yml será:

```
version: "3"

services:
  web:
    container_name: dashboard-web-container-react
    restart: unless-stopped
    build:
      context: .
      dockerfile: ./docker/web/Dockerfile
    command: >
      sh -c "rm -rf celerybeat.pid &&
             python manage.py migrate &&
             python manage.py runserver 0.0.0.0:8585"

    # Lee las variables de entorno del archivo .env
    environment:
      MYSQL_PASSWORD: ${MYSQL_PASSWORD}
      MYSQL_USER: ${MYSQL_USER}
      MYSQL_NAME: ${MYSQL_NAME}
      MYSQL_HOST: ${MYSQL_HOST}
      MYSQL_PORT: ${MYSQL_PORT}
      AEMET_API_KEY: ${AEMET_API_KEY}
      ENVIRONMENT: development
      DJANGO_SECRET_KEY: ${DJANGO_SECRET_KEY}
      DEBUG: 1
    volumes:
      - .:/code
    ports:
      - "9595:8585"
    depends_on:
      - db
      - redis

  db:
    image: mysql
    container_name: dashboard-mysql-react
    restart: unless-stopped
    volumes:
      - mysql-volume-react:/var/lib/mysql
    ports:
      - "13306:3306"
    environment:
      MYSQL_ROOT_PASSWORD: ${MYSQL_PASSWORD}
      MYSQL_DATABASE: ${MYSQL_NAME}

  redis:
    container_name: dashboard-redis-react
    restart: unless-stopped
    image: "redis:alpine"

  celery:
    build:
      context: .
      dockerfile: ./docker/celery/Dockerfile
    container_name: dashboard-celery-react
    restart: unless-stopped
```

```
    environment:
      MYSQL_PASSWORD: ${MYSQL_PASSWORD}
      MYSQL_USER: ${MYSQL_USER}
      MYSQL_NAME: ${MYSQL_NAME}
      MYSQL_HOST: ${MYSQL_HOST}
      MYSQL_PORT: ${MYSQL_PORT}
      AEMET_API_KEY: ${AEMET_API_KEY}
      ENVIRONMENT: development
      DJANGO_SECRET_KEY: ${DJANGO_SECRET_KEY}
      WISE_USER: ${WISE_USER}
      WISE_PASSWORD: ${WISE_PASSWORD}
      WISE_URL_LOGIN: ${WISE_URL_LOGIN}
      WISE_URL_POST: ${WISE_URL_POST}
      WISE_URL_DESTINATION: ${WISE_URL_DESTINATION}
      DEBUG: 1
    command: celery -A dashboard_project worker -l info
    volumes:
      - .:/code
    depends_on:
      - db
      - redis

  celery-beat:
    build:
      context: .
      dockerfile: ./docker/celery/Dockerfile
    container_name: dashboard-celery-beat-react
    restart: unless-stopped
    command: celery -A dashboard_project beat -l INFO --
scheduler django_celery_beat.schedulers:DatabaseScheduler
    environment:
      MYSQL_PASSWORD: ${MYSQL_PASSWORD}
      MYSQL_USER: ${MYSQL_USER}
      MYSQL_NAME: ${MYSQL_NAME}
      MYSQL_HOST: ${MYSQL_HOST}
      MYSQL_PORT: ${MYSQL_PORT}
      AEMET_API_KEY: ${AEMET_API_KEY}
      ENVIRONMENT: development
      DJANGO_SECRET_KEY: ${DJANGO_SECRET_KEY}
      DEBUG: 1
    volumes:
      - .:/code
    depends_on:
      - db
      - redis

volumes:
  mysql-volume-react:
```

Una vez definidos los tres archivos anteriores se ejecuta el siguiente comenda en la consola:

```
>> docker-compose up -d –build
```

Tras esto Docker iniciará cinco contenedores, cada simulará un sistema operativo diferente:

- Contenedor Web: es un contenedor Linux que lleva una imagen Python instalada y que permite desplegar Django fácilmente. En este contenedor se ejecuta tanto el *back-end* como el *front-end.*
- Contenedor db: en el se encuentra la base de datos MySQL
- Contenedor Redis: en el se encuentra la base de datos Redix
- Contenedores Celery: iguales que el contenedor Web, pero se encargan de ejecutar Celery. Esto permite que los scripts del sistema de riego sigan ejecutándose y almacenando valores en la base de datos incluso cuando el servidor web no esté operativo.

La ventaja principal de usar la tecnología Docker es que, ejecutando un simple comando en la consola, se despliega toda la aplicación, independientemente del sistema operativo del ordenador.

Anexo F Conexión con la API de la AEMET

A continuación, se muestra un fragmento del código para conectarse a la API de la AEMET

```
url = ("https://opendata.aemet.es/opendata/api/prediccion/
       especifica/municipio/diaria/")
url_municipio = url + str(codigo_estacion)

headers = {
'cache-control': "no-cache",
'Content-type': 'application/json',
}

response = requests.request(
"GET", url_municipio, headers=headers, params=APIKEY
)

url_datos = response.text.split()[9].replace("\"",
"").strip(",")

url_metadatos = response.text.split()[12].replace("\"",
                  "").strip(",")

if metadatos != "404":
     datos_str = requests.request("GET", url_datos).json()
```

Anexo G Conexión con la base de datos

Para conectarse con la base de datos e insertar los valores en las tablas definidas se ha desarrollado un script en Python. Este código se encarga de conectarse a la base de datos, descargar, insertar o actualizar los valores de las tablas seleccionadas y desconectarse. A continuación, se expone la clase desarrollada:

```
class DatabaseConnect:

    def __init__(self, database, user, password, host,
                 port=3306):
        self.database = database
        self.user = user
        self.password = password
        self.host = host
        self.port = port

    def connect(self):
        try:
            self.con = mysql.connect(host=self.host,
                           port=self.port, user=self.user,
                           password=self.password,
                           database=self.database,
                           charset='utf8mb4',
                           auth_plugin='mysql_native_password')

            self.cur = self.con.cursor()
            print("Base de datos {} abierta
                   exitosamente".format(self.database))

        except Exception as ex:
            print('No se puede conectar con la base de datos')
            print(ex)

    def database_insert(self, sql,single_value=False):

        try:
            if single_value:
                self.connect()
            self.cur.execute(*sql)
            self.con.commit()
            print("Sql insertado exitosamente")
            if single_value:
                self.disconnect()

        except Exception as ex:
            print('No se ha podido conectar con la tabla')
            print('Tipo de error', sys.exc_info())

    def database_select(self, sql, single_value=False):
```

```
        try:
            if single_value:
                self.connect()
            self.cur.execute(sql)
            return self.cur.fetchall()
            print("Sql seleccionado exitosamente")
            if single_value:
                self.disconnect()

        except Exception as ex:
            print('No se ha podido conectar con la tabla')
            print('Tipo de error', ex)

    def disconnect(self):
        try:
            self.con.close()
            print('----Base de datos desconectada----')
        except Exception as ex:
            print('No se ha podido cerrar la base de datos', ex)
```

Anexo H Web Scraping ElTiempo.es

A continuación, se muestra un fragmento del código para obtener los valores de precipitación, porcentaje de nubes y probabilidad de precipitación, haciendo Web Scraping a la web de ElTiempo.es.

```
str1 = "https://www.eltiempo.es/"
str3 = ".html"
url = str1 + estacion_meteorologica + str3

response = get(url)

codigo_html = BeautifulSoup(response.text, 'html.parser')

precipitacion = codigo_html.find_all(
                'div', class_='m_table_weather_day_child
                 m_table_weather_day_rain')

probabilidad_precipitacion = codigo_html.find_all(
               'div', class_="m_table_weather_day_temp_wrapper")

porcentaje_nubes = codigo_html.find_all(
            'div', class_="m_table_weather_day_temp_wrapper")
```

Anexo I Creación del proyecto con Django

Para crear un nuevo proyecto en Django se ejecuta el siguiente comando:

```
>> django-admin startproject dashboard_project
```

Este comando crea una estructura que contendrá una serie de carpetas y ficheros. El árbol de archivos es el siguiente:

```
├── dashboard_project
│   ├── __init__.py
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── manage.py
```

Estos archivos tienen las siguientes funcionalidades:

- *manage.py:* encargado de ejecutar la mayoría de los comandos necesarios para iniciar el servidor, crear las aplicaciones, trabajar con base de datos, etc.
- *settings.py:* contendrá todos los ajustes del *back-end* que se irán definiendo a lo largo del anexo.
- *urls.py:* define las rutas que va a tener la aplicación.
- *wsgi.py*: se usa para que Django se comunique con el servidor

Para acceder al panel de administración es necesario crear un super-usuario escribiendo el siguiente comando:

```
>> python manage.py createsuperuser
```

Django a su vez está formado por aplicaciones. Estas aplicaciones son submódulos que poseen sus propias funcionalidades y que en conjunto forman el proyecto. Se han definido las siguientes Apps:

- accounts: se define todo el modelo de usuarios.

- riego: contiene todo lo relacionado con el cálculo de la programación de riego
- sensores: contiene todo los relacionado con los valores que envían los sensores
- utils: contiene los scripts de Python externos que utiliza la aplicación para ampliar sus funcionalidades.

Para crear una aplicación se introduce el siguiente código en la consola del ordenador:

```
>> python manage.py startapp riego
```

Tras esto Django crea una carpeta con el nombre de la APP que presenta la siguiente estructura:

```
riego
    ├── __init__.py
    ├── admin.py
    ├── apps.py
    ├── migrations
    │   └── __init__.py
    ├── models.py
    ├── tests.py
    └── views.py
```

Por último, hay que añadir dicha aplicación a los ajustes del proyecto en el apartado de aplicaciones instaladas:

Setting.py

```
INSTALLED_APPS = [
…
'sensors.apps.SensorsConfig',
]
```

Anexo J Modelo de usuarios con Django

Para el modelo de usuarios se ha utilizado la librería *django-allauth* que se encarga de los usuarios, grupos, permisos y de manejar las cookies de sesión de usuario. Básicamente, esta librería se encarga de la autenticación, que verifica que el usuario es quien dice que es, y la autorización, que determina que es lo que ese usuario puede ver.

Para instalar la librería se introduce el siguiente código en consola:

```
>> pip install django-allauth
```

Sin embargo, como se explicó en el Anexo E, todas las librerías del proyecto van a estar definidas en el archivo requirements.txt, de esta forma al iniciar el contenedor Docker se instalan todas las que estén definidas.

Una vez se instala la librería es necesario modificar el archivo *setting.py*. Se introduce el nombre de la librería en el apartado de las aplicaciones instaladas:

Setting.py:

```
INSTALLED_APPS = [
      …
    'allauth',
    'allauth.account',
]
```

Tras esto hay que modificar los ajustes de las plantillas y añadir la configuración en el archivo urls.py. Para ello se introduce el siguiente código:

Setting.py:

```
TEMPLATES = [
    {
        'BACKEND':
'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, "templates")],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',

'django.contrib.messages.context_processors.messages',
            ],
```

```
        },
    },
]
```

Urls.py:

```
urlpatterns = [
    # Django admin
    path('administracion-losllanos/', admin.site.urls,),

    # User manageennt
    path('accounts/', include('allauth.urls')),
]
```

Se añade el siguiente *back-end* de configuración:

Setting.py:

```
AUTHENTICATION_BACKENDS = (
    'django.contrib.auth.backends.ModelBackend',
    'allauth.account.auth_backends.AuthenticationBackend',
)
```

Se añade la siguiente configuración para modificar los ajustes por defecto:

Setting.py:

```
# Recuerda el inicio de sesión
ACCOUNT_SESSION_REMEMBER = True
# Redirige a la siguiente URL cuando inicia sesión
LOGIN_REDIRECT_URL = '/app/dashboard/'
# Redirige a la siguiente URL cuando desconecta sesión
ACCOUNT_LOGOUT_REDIRECT_URL = "/accounts/login"
ACCOUNT_AUTHENTICATED_LOGIN_REDIRECTS = True
# No permite email iguales entre usuarios
ACCOUNT_UNIQUE_EMAIL = True
```

Se especifica los ajustes de email para que la aplicación pueda enviar correos a los usuarios en el caso de que estos pidan un cambio de contraseña:

Setting.py:

```
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_USE_TLS = True
EMAIL_PORT = 587
EMAIL_HOST_USER = os.environ.get('EMAIL_HOST_USER')
EMAIL_HOST_PASSWORD = os.environ.get('EMAIL_HOST_PASSWORD')
```

En el código anterior se observa como la variable *EMAIL_HOST_USER y EMAIL_HOST_PASSWORD* son variables de entorno que se encuentran definidas en el archivo *.env.*

Finalmente, una vez definidos todos los ajustes, se procede a crear las plantillas. Inicialmente la librería viene con unas plantillas por defecto que presentan todas las funcionalidades de registro y autenticación de usuario, de modo que únicamente hay que modificar el aspecto visual de las mismas para adaptarlas a la interfaz de la aplicación. A continuación, se muestra la plantilla de inicio de sesión en la que únicamente se le añade la etiqueta *{{form}}* para importar todas las funcionalidades del registro de usuarios:

Login.html:

```
{% extends "account/base.html" %}
{% load crispy_forms_tags%}

{% block content %}
<div class="app-title">
    <h1>Login</h1>
</div>
<div class="login-form">
    <form class="w3-container" method="POST"> {% csrf_token %}
        <div class="control-group">
            {{form|crispy}}
        </div>
        <p>
            <button class="button button2" type="submit">Inciar
           Sesión</button>
            <a class="button secondaryAction" href="{% url
           'account_reset_password' %}">¿Olvidó la
           contraseña?</a>
        </p>

    </form>
</div>
{% endblock %}
```

Se sigue el mismo proceso para las plantillas de desconexión, cambio de contraseña y registro.

Para la parte de cambio de contraseña hay que definir un archivo .txt con el nombre *email_confirmation_subject.txt* que contiene el mensaje que va a enviar el sistema al correo de los usuarios en el caso de que soliciten un cambio de contraseña. En dicho mensaje hay definida una URL que se activa por tiempo limitado para permitir el cambio de clave de manera segura

Anexo K Configuración de la API REST en Django

Una vez instalado el modulo *djangorestframework*, definido en el archivo *requirements.txt*, hay que añadir la siguiente configuración al archivo *settings.py* de Django:

```
INSTALLED_APPS = [
…
'rest_framework',
'corsheaders'
]

REST_FRAMEWORK = {
    # Use Django's standard `django.contrib.auth` permissions,
    # or allow read-only access for unauthenticated users.
    'DEFAULT_PERMISSION_CLASSES': [
        #
'rest_framework.permissions.DjangoModelPermissionsOrAnonReadOnly
'
        # 'rest_framework.permissions.AllowAny',
        'rest_framework.permissions.IsAuthenticated',
    ],

}

CORS_ORIGIN_ALLOW_ALL = True
CSRF_COOKIE_NAME = "XSRF-TOKEN"
```

Anexo L Métodos para obtener los valores de AFA, ADT y Kr

Los valores de AFA, ADT y Kr se calculan a partir de las características de la explotación. Se han creado así dos métodos cuya función es obtener los valores que le llegan del *front-end* y calcular estas variables para posteriormente insertarlas en la base de datos

La formula de la disponibilidad de agua se define en el capítulo 3 del presente proyecto. El método utilizado es el siguiente:

```
def disponibilidad_agua(self, terreno, cultivo, zr):
        ofc = {
            'Arenoso': 0.12,
            'Arenoso-Franco': 0.15,
            'Franco-Arenoso': 0.23,
            'Franco': 0.25,
            'Franco-Limoso': 0.29,
            'Limoso': 0.32,
            'Franco-Arcillo-Limoso': 0.335,
            'Arcillo-Limoso': 0.36,
            'Arcilloso': 0.36,
        }
        owp = {
            'Arenoso': 0.045,
            'Arenoso-Franco': 0.065,
            'Franco-Arenoso': 0.11,
            'Franco': 0.12,
            'Franco-Limoso': 0.15,
            'Limoso': 0.17,
            'Franco-Arcillo-Limoso': 0.205,
            'Arcillo-Limoso': 0.23,
            'Arcilloso': 0.22,
        }
        p = {
            'Almendro': 0.4,
        }
        adt = 1000 * (ofc[terreno] - owp[terreno]) * zr
        afa = adt * p[cultivo]
        return(adt, afa)
```

El método utilizado para calcular el coeficiente Kr, definido en el capítulo 3 del presente proyecto, es el siguiente:

```
    def coeficiente_kr(self, diametro, arboles_ha):
        sc = ((math.pi * (diametro ** 2) * arboles_ha) / 400)
        if sc <= 60:
            kr = round(((2 * sc) / 100), 2)
            return(kr)
        else:
```

```
            return(1)
```

Finalmente, se usa el método *to_internal_value* para rescatar los valores que le llegan del *front-end* y retornar un nuevo objeto con las tres variables calculadas anteriormente

```
def to_internal_value(self, data):
    values = super().to_internal_value(data)
    values['ADT'] = (self.disponibilidad_agua(
        values['Terreno'],
        values['Cultivo'],
        values['Znr'])[0])
    values['AFA'] = (self.disponibilidad_agua(
        values['Terreno'],
        values['Cultivo'],
        values['Znr'])[1])
    values['Kr'] = (self.coeficiente_kr(
        values['Diametro_copa'],
        values['Arboles_ha']))

    print('Se ejecuta do_innternal_value')
    return values
```

Anexo M Configuración de Celery

Una vez definida las librerías *celery, redis, django-celery-beat* y *django-cron*, se añade la siguiente configuración en el archivo *setting.py*:

```
INSTALLED_APPS = [
      …
    'django_cron',
    'django_celery_beat',
]

CELERY_BROKER_URL = 'redis://redis:6379'

CELERY_TIMEZONE = 'Europe/Madrid'
# Let's make things happen
CELERY_BEAT_SCHEDULE = {

    'dr-semanna-every-day-at-6am': {
        'task': 'utils.tasks.dr_semana_celery',
        'schedule': crontab(hour=6),
    },

    'estrategias-every-day-at-6-10am': {
        'task': 'utils.tasks.estrategias_celery',
        'schedule': crontab(hour=6, minute=10),
    },

    'scraping-sensores-every-20minnutes': {
        'task': 'utils.tasks.scraping_joomla_celery',
        'schedule': crontab(minute='*/20'),
    },
}
```

En el código anterior se configura el tiempo para ejecutar las tareas que se definen a continuación. El cálculo de las estrategias se ejecuta todos los días a las 6 AM, mientras que el *script* que descarga los valores de los sensores se ejecuta cada 20 minutos.

Definida la configuración se crea un archivo denominado *tasks.py* dentro de la aplicación *utils*, donde se definen las tareas que va a realizar Celery:

```
@shared_task
def dr_semana_celery():
    os.system("python utils/scripts/dr_semana.py")

@shared_task
def estrategias_celery():
    os.system("python utils/scripts/riego_semana_e1.py")
    os.system("python utils/scripts/riego_semana_e2.py")

@shared_task
def scraping_joomla_celery():
    os.system("python utils/scripts/scraping_sensores.py")
```

Anexo N Archivos que forman el *front-end*

En las siguientes tablas se muestran los archivos que forman el *front-end* de la aplicación

Tabla 7 Plantillas que forman el back-end. Fuente: elaboración propia

PLANTILLAS DE DJANGO	DESCRIPCIÓN.
Front-End	Documento HTML donde React va a desplegar toda la aplicación
Inicio sesión	Documento HTML que muestra un formulario para insertar el usuario y la contraseña
Cerrar sesión	Documento HTML que muestra una ventana para cerrar la sesión
Cambio de contraseña	Documento HTML que muestra un formulario para insertar el email y que se envíe un correo para cambiar la contraseña
Cambio de contraseña enviado	Documento HTML que muestra una página que confirma que se ha enviado un correo para el cambio de contraseña
Cambio de contraseña con clave	Documento HTML que muestra un formulario para introducir la nueva contraseña
Cambio de contraseña realizado	Documento HTML que muestra una página que confirma que el cambio de contraseña se ha efectuado
Registro	Documento HTML que muestra un formulario que permite crear un usuario y una contraseña (por defecto está deshabilitado)

Los scripts encargados del diseño visual son:

Tabla 8 Archivos de estilo CSS3. Fuente: elaboración propia

CSS3	DESCRIPCIÓN.
Registro	Documento HTML que contiene el diseño de la interfaz de registro y autenticación de usuarios
React	Documento HTML que contiene parte del diseño visual de la aplicación en React

Se han utilizado los siguientes scripts en JS para ampliar las funcionalidades.

Tabla 9 Archivos de JavaScript. Fuente: elaboración propia

JavaScript	DESCRIPCIÓN.
Gráficas sensores	Se diseña las gráficas que van a mostrar los valores de los sensores
Gráficas agotamiento	Se diseña las gráficas que van a mostrar los valores referentes al cálculo del agotamiento de humedad en el suelo
Gráficas estrategias	Se diseña las gráficas que van a mostrar los valores referentes al cálculo del riego
Mapa NDVI	Muestra el índice NDVI de la parcela para las fechas dadas

Por ultimo, la parte desarrollada en React se divide en componentes y contenedores. Estos últimos son componentes que simulan una página en la aplicación SPA.

Tabla 10 Contenedores que forman el front-end de la aplicación. Fuente: elaboración propia

Contenedores	DESCRIPCIÓN.
Agotamiento	Muestra una gráfica con el agotamiento de humedad en el suelo para los próximos 7 días
Datos Generales	Muestra los datos de la explotación necesarios para calcular el riego de precisión
Control de Riego	Muestra todos los datos relacionados con el control del riego
Sensores	Muestra en varias gráficas los datos, de humedad y conductividad eléctrica, de los sensores instalados en la finca
Mapa NDVI	Muestra una gráfica con el agotamiento de humedad en el suelo para los próximos 7 días

Tabla 11 Componentes que forman el front-end de la aplicación. Fuente: elaboración propia

Componentes	**DESCRIPCIÓN.**
Barra de navegación superior ordenador	Muestra la barra de navegación de la parte superior en ordenadores
Barra de navegación superior móvil	Muestra la barra de navegación de la parte superior en móviles
Barra navegación lateral ordenador	Muestra la barra de navegación lateral en ordenadores
Agotamiento	Muestra una gráfica con el agotamiento de humedad en el suelo para los próximos 7 días
Estrategia 1	Muestra en una gráfica y una tabla los resultados del cálculo de la estrategia 1
Estrategia 2	Muestra en una gráfica y una tabla los resultados del cálculo de la estrategia 2
Barra navegación lateral móvil	Muestra la barra de navegación lateral en móviles
Alertas	Muestra las alertas
Tablas	Permite visualizar y editar en tablas los valores almacenados en la base de datos
Menú de ajustes	Permite editar características referentes a la interfaz, entre las que se encuentran: • Modo oscuro • Deshabilitar alertas • Elegir periodo de muestra de datos del sensor (1 semana, 1 mes, 1 año o todos) • Deshabilitar gráficas de conductividad eléctrica

Anexo O Ejemplo desarrollo del componente en React

A continuación, se muestra el componente encargado de recibir los ajustes de las gráficas. Este componente está formado por:

- *"todosValores"*: si es *true* se desplegarán todos los datos de la base de datos en las gráficas.
- *"numeroValores"*: despliega el numero de valores definidos si *allValues* es falso
- *"graficasConductividad"*: si es falso las gráficas de conductividad no se despliegan.

Primero se define el estado utilizado *useState* y se le asigna un valor por defecto.

```
const [todosValores, setTodosValores] = React.useState(false);
const [numeroValores, setNumeroValores] = React.useState(1500);
const [graficasConductividad, setGraficasConductividad] =
React.useState(false);
```

Luego se crea *useEffect* que se ejecuta cuando se accede por primera vez al componente (se accede a la aplicación) y cuando se actualiza (cambia de valor).

```
useEffect(() => {
   axios.get(`${URL}/api/sensores/setting/data_number/`)
     .then(res => {
       const data = res.data
       const settingData = data[0]
       setTodosValores(settingData.Todos)
       setNumeroValores(settingData.Numero)
       setGraficasConductividad(settingData.Conductividad)
     })
     .catch(error => {
       console.log('Error en ajustes graficas =  ',
                   error.response)
     });

 }, [])
```

En el código anterior se utiliza la librería *Axios* para acceder a la API REST y comunicarse con el *back-end*. Esta librería permite hacer operaciones como cliente HTTP,

enviando solicitudes al servidor donde se encuentra *el back-end* y recibiendo las respuestas con la información solicitada.

Finalmente se crea un botón que ejecuta una función. Esta función actualiza la base de datos alojada en el *back-end* y cambia el estado del componente. A continuación, se muestra la función que cambia el valor de *"todosValores"* a *true* o *false.* De este modo cada vez que se pulsa el botón, las gráficas mostrarán todos los valores de la base de datos o solo aquellos definidos en *"numeroValores"*

```
function cambiarTodosValores() {
    axios
      .put(`${URL}/api/sensores/setting/data_number/1/`,
        { "Numero": numeroValores, "Todos": !todosValores,
                "Conductividad": graficasConductividad }
      )
      .then(res => {
          // se cambia el estado de todosValores
        setNumeroValores(!numeroValores)
      })
      .catch(error => {
        console.log('error al insertar datos en la API REST =
                    ', error.response)
      });
  }
```

Anexo P Configuración del servidor web NGINX

Para pasar la aplicación de modo desarrollo a modo producción es necesario configurar un servidor que se encargue de ello. Para ello se va a crear un nuevo fichero, denominado *docker-compose-prod.yml*, en el que se añade el contenedor de Nginx:

```
...
nginx:
    build: ./nginx
    container_name: dashboard-nginx-react
    restart: unless-stopped
    cap_drop:
      - ALL
    cap_add:
      - chown
      - net_bind_service
      - setgid
      - setuid
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - static-volume:/code/staticfiles
    depends_on:
      - web
```

Para configurar el servidor es necesario modificar el archivo *nginx.conf* y añadir la siguiente configuración:

```
upstream dashboard_project {
    server web:8000;
}
server {
    listen 80;
    server_name losllanosriego.com;
    rewrite ^ https://$http_host/ ;
}

server {
    listen 443 ssl;
    server_name losllanosriego.com;


    ssl_certificate
     /etc/letsencrypt/live/losllanosriego.com/fullchain.pem;
    ssl_certificate_key
     /etc/letsencrypt/live/losllanosriego.com/privkey.pem; #
     managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed
     by Certbot
```

```
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by
     Certbot

    location / {
        proxy_pass http://dashboard_project;
        proxy_set_header X-Forwarded-For
          $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto https; # new
        proxy_set_header Host $host;
        proxy_redirect off;
    }

    location /staticfiles/ {
        alias /code/staticfiles/;
    }
}
```

En la configuración anterior se referencia la carpeta *staticfiles* donde se encuentran todos los archivos estáticos del proyecto y se configura el certificado SSL, obtenido de la autoridad de certificación Let's Encrypt.

Anexo Q Mapa NDVI desarrollado en Google Earth Engine

Se ha desarrollado un script en JS que se ejecuta en la consola de Google Earth Engine y crea una aplicación que muestra un mapa de NDVI de las fechas seleccionadas

```
// Se crea el mapa principal
var mapPanel = ui.Map();

// Se define el punto inicial
var initialPoint = ee.Geometry.Point -4.587476, 37.872162);
mapPanel.centerObject(initialPoint, 9);

// Se crea el panel
var panel = ui.Panel({style: {width: '30%'}});

// Se añade una lista desplegable para los años
var actualyear = fecha.getFullYear();
var i = 2015;
var year = [];
while (i<=actualyear){
  year.push(i.toString());
  i = i+1
}

//var year = ['2023','2022','2021','2020', '2019', '2018',
'2017','2016','2015'];

// Se añade una lista desplegable para los meses y días
var month = ['1','2', '3', '4','5','6','7', '8',
'9','10','11','12'];
var day = ['1','2', '3', '4','5','6','7', '8',
'9','10','11','12','13', '14','15','16','17', '18',
'19','20','21','22', '23', '24','25','26','27', '28',
'29','30','31'];

var selectYear1 = ui.Select({
  items: year,
  value: year[year.length - 1],
  style:{padding: '0px 0px 0px 10px', stretch: 'horizontal'}
});
var selectYear2 = ui.Select({
  items: year,
  value: year[year.length - 1],
  style:{padding: '0px 0px 0px 10px', stretch: 'horizontal'}
});

var selectMonths1 = ui.Select({
  items: month,
  value:month[0],
  style: {stretch: 'horizontal'}
});
var selectMonths2 = ui.Select({
  items: month,
```

```
  value:month[1],
  style: {stretch: 'horizontal'}
});
var selectDay1 = ui.Select({
  items: day,
  value: day[0],
  style:{padding: '0px 0px 0px 10px', stretch: 'horizontal'}
});
var selectDay2 = ui.Select({
  items: day,
  value: day[0],
  style:{padding: '0px 0px 0px 10px', stretch: 'horizontal'}
});


// Opciones de la gráfica
var options = {
      title: 'NDVI',
      vAxis: {title: 'valor ndvi'},
      hAxis: {
      title: 'fecha',
      format: 'dd-MM',

      }
    };

var visparamNDVI =  {
      min:0, max:1,
      palette : [
      'FFFFFF', 'CE7E45', 'DF923D', 'F1B555', 'FCD163',
      '99B718', '74A901', '66A000', '529400', '3E8601', '207401',
      '056201', '004C00', '023B01', '012E01', '011D01',
      '011301']}
//Se crea la paleta de colores
function makeColorBarParams(palette) {
  return {
    bbox: [0, 0, 1, 0.1],
    dimensions: '100x10',
    format: 'png',
    min: 0,
    max: 1,
    palette: palette,
  };
}
// Se crea la barra de color para la leyenda
var colorBar = ui.Thumbnail({
  image: ee.Image.pixelLonLat().select(0),
  params: makeColorBarParams(visparamNDVI.palette),
  style: {stretch: 'horizontal', margin: '0px 8px', maxHeight:
'24px'},
});

// Se crea un panel
var legendLabels = ui.Panel({
  widgets: [
    ui.Label(visparamNDVI.min, {margin: '4px 8px'}),
```

```
    ui.Label(
        (visparamNDVI.max / 2),
        {margin: '4px 8px', textAlign: 'center', stretch:
        'horizontal'}),
    ui.Label(visparamNDVI.max, {margin: '4px 8px'})
  ],
  layout: ui.Panel.Layout.flow('horizontal')
});

var legendTitle = ui.Label({
  value: 'Rango de colores de NDVI',
  style: {fontWeight: 'bold'}
});

function NDVI_CHART(entrada){
        var ndvi =
           entrada.addBands(
           entrada.normalizedDifference(['B8','B4']).
          rename('NDVI'))
        return (ndvi)
        }
var today = (fecha.getDate()+'-'+(fecha.getMonth()+1)+'-
'+fecha.getFullYear()).toString()
function validate_fechaMayorQue(fechaInicial,fechaFinal)
        {
            var valuesStart=fechaInicial.split("-");
            var valuesEnd=fechaFinal.split("-");

            // Verificamos que la fecha no sea posterior a la
                actual
            var dateStart= new
                Date(valuesStart[2],(valuesStart[1]-
                1),valuesStart[0]);
            var dateEnd = new Date(valuesEnd[2],(valuesEnd[1]-
                                1),valuesEnd[0]);
            if(dateStart>=dateEnd)
            {
                return 0;
            }
            return 1;
        }


// Add button to run
var runButton = ui.Button({
  label: 'Calculate extent',
  onClick: function(){
    var fecha1 = (selectDay1.getValue()+"-" +
                  selectMonths1.getValue()+
                   "-" + selectYear1.getValue());
    var fecha2 = (selectYear2.getValue()+"-"+
                  selectMonths2.getValue()+
                   "-"+selectDay2.getValue());

    if(validate_fechaMayorQue(today,fecha1)){
```

```
        alert("La fecha "+fecha1+" es superior a la fecha de
                hoy: "+today+". Introduzca una fecha correcta
                     ");
    }
    else if(validate_fechaMayorQue(fecha1,'01-07-2015')){
       alert("La fecha "+fecha1+" es inferior a la fecha 01-11-
                2015 . Introduzca una fecha mayor ");
    }

    else {

      var collection = ee.ImageCollection ('COPERNICUS/S2')
        .filterDate(selectYear1.getValue()+
        "-"+ selectMonths1.getValue()+"-"+
        selectDay1.getValue(),selectYear2.getValue()+"-"+
        selectMonths2.getValue()+"-"+selectDay2.getValue())
        .filterBounds(geometry)
        .filterMetadata ('CLOUDY_PIXEL_PERCENTAGE', 'Less_Than',
                         10);

      var SentinelFiltro = ee.Image(collection.mean());
      var SentinelClip = SentinelFiltro.clip(geometry);
      var ndvi = SentinelFiltro.normalizedDifference(['B8',
                'B4']);
      var NdviClip = ndvi.clip(geometry);
      mapPanel.centerObject (SentinelClip,14);
      mapPanel.addLayer(NdviClip,
        visparamNDVI,
         selectDay1.getValue()+ "-" + selectMonths1.getValue()+
          "-" + selectYear1.getValue(),
           1);


      var coleccion_ndvi = collection.map(NDVI_CHART);
      var serie_media =
                     ui.Chart.image.series(
                     coleccion_ndvi.select('NDVI'),
                        geometry, ee.Reducer.mean(),
                           20).setOptions(options);

      panel.widgets().set(5, serie_media);
      var legendPanel = ui.Panel([legendTitle, colorBar,
                         legendLabels]);
      panel.widgets().set(6, legendPanel);
    }
  },


      style: {padding: '0px 10px', stretch: 'horizontal'}

});

// Se crea el panel
var panel = ui.Panel({
  layout: ui.Panel.Layout.flow('vertical'),
  style: {width: '400px'}
```

```
});

// Añadir el titulo
var mapTitle = ui.Label('NDVI EXPLOTACION AGRARIA');
mapTitle.style().set('color', 'green');
mapTitle.style().set('fontWeight', 'bold');
mapTitle.style().set({
  fontSize: '20px',
  padding: '10px'
});


// Estilo del cursor
Map.style().set('cursor', 'crosshair');

panel.add(mapTitle);
panel.add(mapDesc);
panel.add(ui.Panel([selectYear1, selectMonths1, selectDay1],
                   ui.Panel.Layout.flow('horizontal')));
panel.add(ui.Panel([selectYear2, selectMonths2, selectDay2],
                   ui.Panel.Layout.flow('horizontal')));
panel.add(runButton);
ui.root.insert(0, panel);

// Se crea el mapa principal y se selecciona la capa SST
ui.root.clear();
ui.root.add(ui.SplitPanel(panel, mapPanel));
```