



**Universidad  
Rey Juan Carlos**

**GRADO EN INGENIERÍA EN TELEMÁTICA**

Curso Académico 2021/2022

Trabajo Fin de Grado

**DESARROLLO DE UN SISTEMA DE  
AUTOMATIZACIÓN Y MONITORIZACIÓN DE  
UN INVERNADERO**

Autor : Víctor Manuel Rincón Yepes

Tutor : David Roldán Álvarez



# Trabajo Fin de Grado

# Desarrollo de un sistema de automatización y monitorización de un invernadero

**Autor :** Víctor Manuel Rincón Yepes

**Tutor :** David Roldán Álvarez

La defensa del presente Proyecto Fin de Carrera se realizó el día \_\_\_\_\_ de \_\_\_\_\_ de 202X, siendo calificada por el siguiente tribunal:

## Presidente:

## **Secretario:**

## Vocal:

y habiendo obtenido la siguiente calificación:

## **Calificación:**

Fuenlabrada, a de de 202X

*Dedicado a  
mi familia*

# **Agradecimientos**

Aquí vienen los agradecimientos... Aunque está bien acordarse de la pareja, no hay que olvidarse de dar las gracias a tu madre, que aunque a veces no lo parezca disfrutará tanto de tus logros como tú... Además, la pareja quizás no sea para siempre, pero tu madre sí.

# **Resumen**

Un resumen

# **Summary**

Un resumen en inglés

# Índice general

<b>Summary</b>	<b>IV</b>
<b>1. Introducción</b>	<b>1</b>
1.1. Internet de las Cosas . . . . .	1
1.2. Tecnologías web . . . . .	3
1.3. Jardines autónomos . . . . .	3
1.4. Estructura de la memoria . . . . .	3
<b>2. Objetivos</b>	<b>4</b>
2.1. Objetivo general . . . . .	4
2.2. Objetivos . . . . .	4
2.3. Planificación temporal . . . . .	5
2.4. Plan de trabajo . . . . .	5
<b>3. Herramientas</b>	<b>7</b>
3.1. Hardware . . . . .	7
3.1.1. Raspberry Pi . . . . .	7
3.1.2. Arduino . . . . .	9
3.1.3. Sensores . . . . .	11
3.1.4. Actuadores . . . . .	13
3.2. Software . . . . .	14
3.2.1. Raspbian . . . . .	14
3.2.2. Python . . . . .	14
3.2.3. Bash . . . . .	15
3.2.4. Dash . . . . .	15

## **ÍNDICE GENERAL**

---

3.2.5. API de telegram . . . . .	17
3.2.6. MongoDB . . . . .	17
3.2.7. Cron . . . . .	17
3.2.8. Docker . . . . .	18
3.2.9. Heroku . . . . .	18
3.2.10. Control de versiones . . . . .	19
<b>4. Diseño e implementación</b>	<b>21</b>
4.1. Montaje físico del invernadero . . . . .	21
4.2. Arquitectura e implementación hardware . . . . .	23
4.3. Arquitectura e implementación software . . . . .	26
4.3.1. Arquitectura general . . . . .	26
4.3.2. Recogida de datos . . . . .	29
4.3.3. Aplicación de Dash . . . . .	31
4.3.4. Bot de Telegram . . . . .	34
4.3.5. Aplicación de riego . . . . .	38
<b>5. Casos de uso</b>	<b>41</b>
5.1. caso dedato bueno cambiar nombre . . . . .	41
5.1.1. Temperatura y humedad ambiente . . . . .	41
5.1.2. Temperatura de la tierra . . . . .	49
5.1.3. Humedad de la tierra . . . . .	53
5.2. cosas particulares cambiar nombre . . . . .	56
<b>6. Conclusiones</b>	<b>60</b>
6.1. Conclusiones finales . . . . .	60
6.2. Trabajos futuros . . . . .	60
<b>A. Manual de usuario</b>	<b>61</b>
<b>Bibliografía</b>	<b>62</b>

# Índice de figuras

3.1.	Raspberry Pi 3B+ . . . . .	8
3.2.	Diagrama de conexiones de Raspberry Pi Model 3B+ . . . . .	9
3.3.	Diagrama de pines de Raspberry Pi Model 3B+ . . . . .	9
3.4.	Arduino UNO rev3 . . . . .	10
3.5.	Diagrama de conexiones y pines de Arduino UNO rev3 . . . . .	11
3.6.	Módulo DHT22 . . . . .	12
3.7.	Sonda DS18B20 . . . . .	12
3.8.	Módulo YL-69 . . . . .	12
3.9.	Bomba de agua modelo CESFONJER . . . . .	13
3.10.	Módulo de relés . . . . .	13
3.11.	Página de ejemplo sin estilo . . . . .	16
3.12.	Página de ejemplo con estilo CSS . . . . .	16
3.13.	Diagrama simplificado del funcionamiento de la virtualización por contenedores . . . . .	18
3.14.	Esquema de concepto básico de Git . . . . .	20
4.1.	Implementación física del invernadero . . . . .	21
4.2.	Bomba de agua en el depósito . . . . .	22
4.3.	Aquí va una imagen de las salidas de agua . . . . .	22
4.4.	Caja de registro abierta . . . . .	23
4.5.	Caja de registro cerrada . . . . .	23
4.6.	Conexiones internas Protoboard . . . . .	24
4.7.	Conexiones internas Protoboard . . . . .	25
4.8.	Esquema de conexionado para un actuador . . . . .	25
4.9.	Esquema de la implementación software del sistema . . . . .	27

## ÍNDICE DE FIGURAS

---

4.10. Esquema general de la estructura de la base de datos. . . . .	28
4.11. Estructura de la colección <i>sensors_data</i> , perteneciente a la base de datos <i>greenhouseDB</i> . . . . .	28
4.12. Estructura de la colección <i>controllers_data</i> , perteneciente a la base de datos <i>greenhouseDB</i> . . . . .	28
4.13. Aplicación de visualización de datos. . . . .	32
4.14. Aplicación de visualización de datos. . . . .	33
4.15. Cambio tras selección de fecha y sensor en la humedad ambiente. . . . .	33
4.16. Mensaje de error enviado cuando el proceso de obtención de la imagen o el envío de esta falla. . . . .	37
4.17. Mensaje informativo recibido tras la activación del riego. . . . .	39
4.18. Mensaje de posible error enviado cuando la diferencia entre dos sensores de humedad de la tierra es significativa. . . . .	39
5.1. Datos almacenados en <i>sensors_data</i> para la temperatura ambiente. . . . .	42
5.2. Datos almacenados en <i>sensors_data</i> para la humedad ambiente. . . . .	42
5.3. Gráfica obtenida mediante la aplicación web con datos de temperatura ambiente desde 1 al 8 de Junio de 2022. . . . .	43
5.4. Uso del selector de fechas en la aplicación web. . . . .	43
5.5. Uso del selector de sensores en la aplicación web. . . . .	43
5.6. Gráfica obtenida mediante el bot de Telegram con datos de temperatura ambiente desde 1 al 8 de Junio de 2022. . . . .	44
5.7. Gráfica obtenida mediante el comando <i>/ambienttemperaturefig</i> realizado el día 12 de Junio de 2022. . . . .	45
5.8. Gráfica obtenida mediante el comando <i>/ambienttemperaturefig</i> realizado el día 12 de Junio de 2022. . . . .	45
5.9. Gráfica obtenida mediante el comando <i>/ambienttemperaturefig</i> realizado el día 12 de Junio de 2022. . . . .	46
5.10. Mensaje de error obtenido el 11 de Junio de 2022 tras error de lectura de temperatura ambiente del sensor 27. . . . .	46
5.11. Mensaje recibido haciendo uso del comando <i>/ambienttemperature</i> . . . . .	47

## ÍNDICE DE FIGURAS

---

5.12. Mensaje recibido haciendo uso del comando <i>/ambienttemperaturedb</i> . . . . .	47
5.13. Mensaje recibido haciendo uso del comando <i>/sensorsinfo</i> . . . . .	47
5.14. Gráfica obtenida mediante la aplicación web con datos de temperatura ambiente desde 1 al 8 de Junio de 2022. . . . .	48
5.15. Gráfica obtenida mediante el bot de Telegram con datos de temperatura ambiente desde 1 al 8 de Junio de 2022. . . . .	48
5.16. Mensaje recibido haciendo uso del comando <i>/ambienthumidity</i> . . . . .	49
5.17. Mensaje recibido haciendo uso del comando <i>/ambienthumiditydb</i> . . . . .	49
5.18. Arquitectura de los datos almacenados en <i>sensors_data</i> para la temperatura de la tierra. . . . .	49
5.19. Sección <i>GROUND TEMPERATURE</i> en la aplicación web de Dash . . . . .	50
5.20. Gráfica obtenida mediante la aplicación web de Dash con datos de temperatura de la tierra desde 1 al 8 de Junio de 2022 . . . . .	50
5.21. Mensaje recibido haciendo uso del comando <i>/groundtemperature</i> . . . . .	51
5.22. Mensaje recibido haciendo uso del comando <i>/groundtemperaturedb</i> . . . . .	51
5.23. Diferencia entre lecturas de ambos sensores. Medidas tomadas entre las 20 del dia 11 de Junio de 2022 y las 14 del día 12 de Junio de 2022 . . . . .	51
5.24. Gráfica obtenida mediante el comando <i>/groundtemperaturefig</i> realizado el día 12 de Junio de 2022. . . . .	52
5.25. Gráfica obtenida mediante el bot de Telegram con datos de temperatura ambiente desde 1 al 8 de Junio de 2022. . . . .	52
5.26. Arquitectura de los datos almacenados en <i>sensors_data</i> para la humedad de la tierra. . . . .	53
5.27. Gráfica con datos de riego del 4 al 6 de junio de 2022. . . . .	54
5.28. Gráfica con datos de humedad de la tierra del 4 al 6 de junio de 2022. . . . .	54
5.29. Datos de riego obtenidos mediante la aplicación web del 04 al 07 de junio de 2022. . . . .	55
5.30. Datos de riego obtenidos mediante el bot de Telegram del 04 al 07 de junio de 2022. . . . .	55
5.31. Salida del comando <i>/controllersinfo</i> . . . . .	56
5.32. Gráfica que muestra un comportamiento anormal de uno de los higrómetros. . .	56

## **ÍNDICE DE FIGURAS**

---

5.33. Gráfica que muestra un comportamiento anormal de uno de los sensores de temperatura ambiente. . . . .	56
5.34. Salida del comando <i>/sensorsdata</i> . . . . .	57
5.35. Salida del comando <i>/sensorsdataadb</i> . . . . .	58
5.36. Primera parte salida del comando <i>/getallfigures</i> . . . . .	59
5.37. Segunda parte salida del comando <i>/getallfigures</i> . . . . .	59
5.38. Tercera parte salida del comando <i>/getallfigures</i> . . . . .	59

# Capítulo 1

## Introducción

Una introducción

### 1.1. Internet de las Cosas

El Internet de las cosas, *Internet Of Things* (IoT), en inglés es un concepto que se refiere a la conexión vía Internet de elementos y objetos físicos de la vida cotidiana de manera que se permita el envío y la recepción de diferentes datos.

Uno de los primeros hitos que marcan el nacimiento de este concepto data de 1874. Con el objetivo de realizar estudios de meteorológicos un grupo de científicos franceses instalaron en la cima del Mont-Blanc una serie de sensores para medir temperaturas, vientos y humedades, entre otros. Dicha información sería enviada por radio a París.

Nikola Tesla, en 1926 llegó a hablar de una interconexión de todo en un ámbito global, lo que él denominaba un *gran cerebro*. Esto ocurrió mucho antes de la llegada de Internet tal y como lo conocemos hoy en día. Cabe destacar también a Alan Turing, ya que muchas de sus publicaciones se consideran base del IoT, Turing afirmaba que existía una necesidad de aportar capacidades de comunicación a sensores.

Otro de los puntos históricos claves para la llegada del IoT es el nacimiento de Internet. En el año 1969 nacería ARPANET, como continuación del proyecto ARPA (Advanced Research Projects Agency), que nacería en 1957 y pretendía proponer avances tecnológicos y militares. ARPA consiguió múltiples avances como un sistema de comunicaciones en el que si una máquina caía, el resto podía seguir conectadas. En 1965 se consiguió comunicar en Estados

## **1.1. INTERNET DE LAS COSAS**

---

Unidos un ordenador TX2 en Massachusetts con un Q-32 Wn California mediante una línea telefónica conmutada. El proyecto ARPA sigió investigando en esta línea hasta que en 1969 con la incorporación del investigador Michel Elie a UCLA (University of California, Los Angeles) y a ARPA se consiguió conectar dos máquinas, una de la UCLA con otra del SRI (Stanford Research Institute), poco después serían cuatro ordenadores distribuidos en diferentes universidades de Estados Unidos, en una red que sería denominada ARPANET. En 1972 eran 50 las universidades y centros de investigación que estaban conectados a ARPANET. Cuando el Departamento de Defensa de Estados Unidos decidió 1983 usar los protocolos TCP/IP nacería ARPA Internet, que más tarde pasó a conocerse como Internet. En 1991, un año después de que Tim Berners Lee desarrollara HTML y HTTP, el Internet que conocemos hoy en día nació en el CERN (European Organization for Nuclear Research). En agosto de 1991, cualquiera podría acceder a esta red.

No es hasta el año 2009 que, de la mano del profesor del MIT (Massachusetts Institute of Technology) Kevon Ashton surge el término *Internet de las Cosas*.

Con el paso de los años y la combinación de distintas tecnologías y campos de estudio como el Big Data, el Machine Learning o la Inteligencia Artificial el IoT se ha ido asentando y creciendo dando lugar incluso a nuevos conceptos como las Smart Cities o Ciudades inteligentes, una aplicación del IoT junto a estas tecnologías.

El IoT está en continuo crecimiento, según Cisco Internet Solutions Group (IBSG) entre los años 2008 y 2009 ya eran más los objetos conectados a Internet que las personas y se estima que para el año 2025 habrá unos 75 mil millones de objetos conectados a Internet. Como se ha comentado anteriormente, la tendencia actual del IoT, es la del tratado masivo de datos, la automatización de procesos y la toma de decisiones en función de los múltiples datos recolectados.

En este TFG se ha implementado un invernadero autónomo haciendo uso de esta tecnología, aplicando soluciones IoT. En 2017 se invirtieron 10.000 millones de dólares en la digitalización de la agricultura. La toma de decisiones basada en los datos tomados por los diferentes tipos de sensores en la agricultura es la tendencia actual, un invernadero conectado y con estas capacidades permite una producción más ecológica y sostenible, reduce el consumo de agua necesaria para los cultivos así como la necesidad del uso de productos químicos. Según la Organización de las Naciones Unidas para la Alimentación y la Agricultura(FAO), el sector agrícola debería producir un 50 % más para poder cubrir la demanda para el año 2050, la falta de inversión y di-

## **1.2. TECNOLOGÍAS WEB**

---

gitalización de este sector podría poner en riesgo nuestro sistema de alimentación. La inversión y la investigación en IoT puede aportar una mejora importante en la digitalización y ayudar así a combatir este problema.

### **1.2. Tecnologías web**

### **1.3. Jardines autónomos**

### **1.4. Estructura de la memoria**

# **Capítulo 2**

## **Objetivos**

### **2.1. Objetivo general**

En este capítulo se explican los objetivos que se pretenden conseguir en este trabajo de fin de grado, la planificación para su consecución y la metodología seguida.

### **2.2. Objetivos**

Este TFG consiste en crear un sistema distribuido que permita monitorizar y automatizar un invernadero. Para ello se han planteado varios subobjetivos:

- Desarrollo de un entorno de bajo costo que permita realizar la recolección y almacenamiento de datos a partir de sensores. Trabajando con magnitudes muy utilizadas en el cuidado de cualquier invernadero, en concreto, humedad ambiente, temperatura ambiente, humedad de la tierra y temperatura de la tierra.
- Desarrollo de una aplicación web cuyo objetivo sea mostrar gráficos en función de los datos recogidos.
- Desarrollo de un bot de Telegram que permita consultar datos tanto en tiempo real como desde la base de datos, además de otras funcionalidades como la consulta de gráficas sobre los datos recogidos o el envío de notificaciones al usuario cuando es pertinente, como errores en las lecturas, posibles fallos o acciones tomadas por los actuadores implementados.

## **2.3. PLANIFICACIÓN TEMPORAL**

---

- Desarrollo de scripts que permitan automatizar el riego del invernadero de manera inteligente, usando la información obtenida por los sensores de humedad de la tierra.

## **2.3. Planificación temporal**

Para desarrollar este TFG se ha seguido una metodología *Scrum* basada en *sprints*. *Scrum* es una metodología de desarrollo de software ágil. Esta nace en contraposición al llamado desarrollo *waterfall*, o desarrollo en cascada. Un *sprint* es un ciclo de trabajo, suelen durar entre 1 y 4 semanas. En este caso en concreto, las reuniones han sido semanales, en estas se ha presentado la última versión del proyecto al tutor y, además, se han concretado los objetivos para el siguiente ciclo.

Como se expone en el capítulo 3, para el control de versiones se han utilizado las herramientas Git y Github. Para la resolución de incidencias concretas se han creado *issues* que se han resuelto en una rama derivada de *master*. Una vez resueltas, este código ha sido incorporado a la rama *master* mediante una *Pull request*.

## **2.4. Plan de trabajo**

Las etapas en las que se ha dividido el desarrollo del proyecto para así conseguir los objetivos anteriormente mencionados han sido:

- Estudio previo del tema del TFG, leyendo diferente literatura sobre automatización y monitorización de jardines e invernaderos.
- Diseño del sistema distribuido a nivel software y hardware, decidiendo además cuales de los componentes hardware disponibles en el mercado son los más adecuados para la implementación final.
- Implementación electrónica de los sensores y desarrollo de los scripts de testeo para estos, recolección de datos y subida a la base de datos de MongoDB. En primer lugar se implementaron los sensores de humedad y temperatura ambiente y una vez se probó el correcto funcionamiento del sistema, se añadieron los de humedad y temperatura de la tierra.

## *2.4. PLAN DE TRABAJO*

---

- Desarrollo de la aplicación web para mostrar los datos disponibles y crear figuras a partir de estos.
- Desarrollo del bot de Telegram para poder obtener datos en tiempo real, desde la base de datos y generar gráficas a partir de estos.
- Inclusión de los actuadores en el sistema y desarrollo de los scripts de automatización del invernadero. Añadiento, además, soporte para los actuadores en la aplicación web y en el bot de Telegram.
- Añadir mayor soporte para el control de errores creando scripts capaces de notificar al usuario vía Telegram sobre estos. A pesar de que las etapas están claramente diferenciadas, en todos los puntos se ha ido mejorando el control de errores y resolviendo *bugs* que se pudieran detectar.

En cada una de las etapas hubo una gran parte de inciación e investigación sobre las tecnologías utilizadas, por ejemplo, para la aplicación web, se crearon diferentes programas de prueba para comprender los conceptos básicos del framework utilizado, *Dash*.

# **Capítulo 3**

## **Herramientas**

En este capítulo se describen de las distintas herramientas utilizadas para este TFG. Las diferentes tecnologías que se mostrarán en este capítulo se pueden dividir en hardware, incluyendo tanto los sensores y actuadores como la Raspberry pi y el Arduino, y software, en este último se concretarán las herramientas principales de desarrollo así como las utilizadas para el almacenamiento de los datos recogidos por los sensores.

### **3.1. Hardware**

A continuación se presenta el hardware utilizado, todas estas tecnologías utilizadas son de bajo coste, y entre las múltiples posibilidades existentes en el mercado se han elegido las siguientes para abaratar el presupuesto necesario para realizar este TFG.

#### **3.1.1. Raspberry Pi**

Para el desarrollo del TFG se ha utilizado una Raspberry Pi modelo 3B+ (ver figura 3.1). Se trata de un ordenador de placa única concebido por la fundación británica Raspberry Pi Foundation y cuyo primer modelo fue lanzado el 29 de febrero de 2012. En primera instancia, este microcomputador fue desarrollado para acercar la programación y la informática al ámbito educativo. Rápidamente alcanzó una gran popularidad y gracias a la buena acogida está altamente extendido para múltiples proyectos de diferente índole, entre estos proyectos destacan diferentes campos como pueden ser el de la robótica, IoT o la domótica.

### 3.1. HARDWARE

---

Algunas de las principales características de la Raspberry Pi 3 Model B+ son:

- Procesador BroadcomBCM2837B0, Cortex-A53(ARMv8) 64-bit SoC
- Frecuencia de reloj de 1,4 GHz.
- Memoria RAM de 1 GB LPDDR2 SDRAM.
- Bluetooth 4.2 BLE
- Wi-Fi Dual Band b/g/n/ac
- Gigabit Ethernet
- Header GPIO 40 pines.
- Cuatro puertos USB 2.0, puerto de cámara CSI, puerto de visualización DSI y puerto HDMI

Estas características, junto a su ajustado precio, han sido motivo para elegir este modelo.

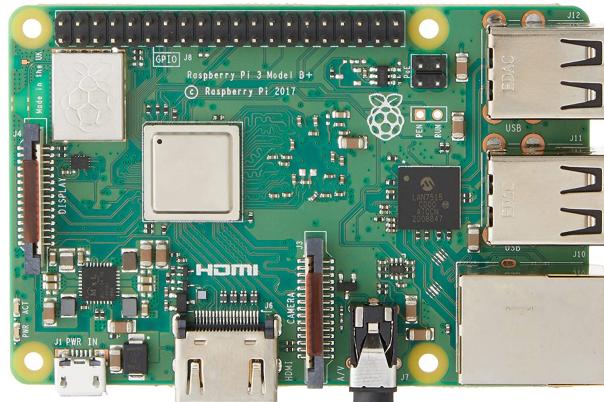


Figura 3.1: Raspberry Pi 3B+.

La Raspberry Pi posee una serie de entradas y salidas, como se muestra en la figura 3.2. A lo largo del desarrollo de la memoria se utiliza la nomenclatura BCM que es la utilizada por el procesador de la Raspberry (ver figura 3.3). Como se puede apreciar, la placa tiene un total de 40 pines GPIO con diferentes propósitos entre los que destacan pines con entradas de 3,3 y 5V, pines a tierra y pines de propósito general (GPIO).

### 3.1. HARDWARE

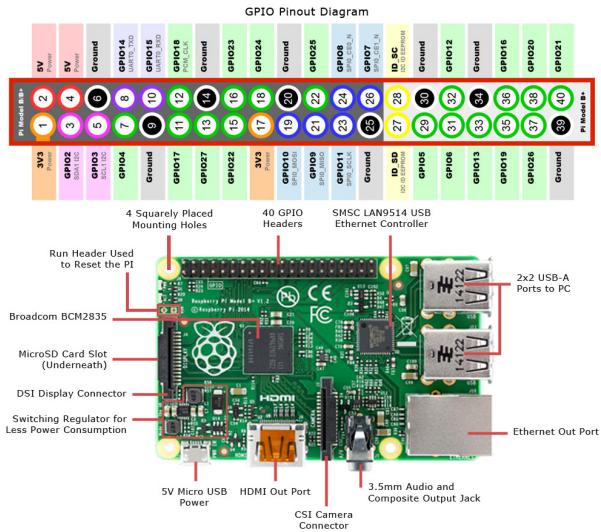


Figura 3.2: Diagrama de conexiones de Raspberry Pi Model 3B+.

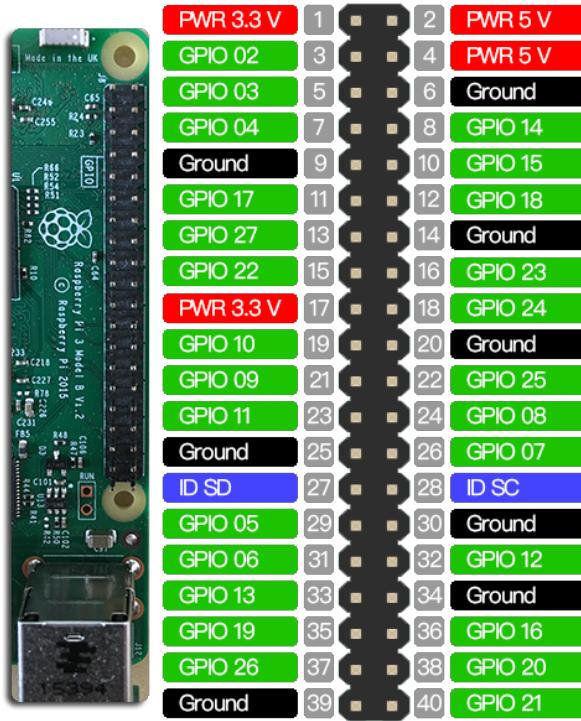


Figura 3.3: Diagrama de pines de Raspberry Pi Model 3B+.

#### 3.1.2. Arduino

Arduino es un proyecto de software libre que, así como el de Raspberry Pi, también fue desarrollado con fines educativos. Está basado en diferentes placas con un microcontrolador.

### 3.1. HARDWARE

---

Para trabajar con Arduino se suele trabajar utilizar un entorno de programación específico para dichas placas. El lenguaje de programación que se suele utilizar es una adaptación de C++, un lenguaje orientado a objetos, de alto nivel y multiparadigma, pero en este caso se ha utilizado la librería Pyfirmata2, que implementa el protocolo para microcontroladores Firmata. Dicha librería permite interactuar con la placa Arduino usando Python, lenguaje de programación con el que se ha desarrollado la mayoría de este TFG y que se indica en la sección software.

Se ha utilizado la placa Arduino UNO rev3 (ver figura 3.4), cuyas principales características son:

- Microcontrolador ATmega328P.
- Funcionamiento a 5V.
- Voltaje de entrada de 6 a 20V.
- 14 pines digitales, 6 de ellos con salida PWM.
- 6 pines de entrada analógica.

De nuevo su precio ha sido motivo para la elección de este modelo de microcontrolador.

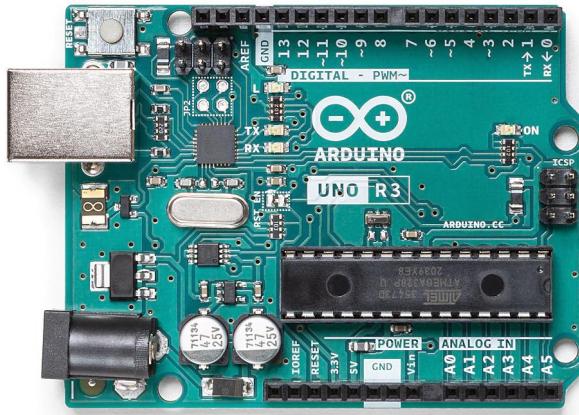


Figura 3.4: Arduino UNO rev3.

El Arduino cuenta con una serie de conexiones que son especialmente relevantes para la implementación que se ha llevado a cabo en este TFG que se muestra en la figura 3.5

### 3.1. HARDWARE

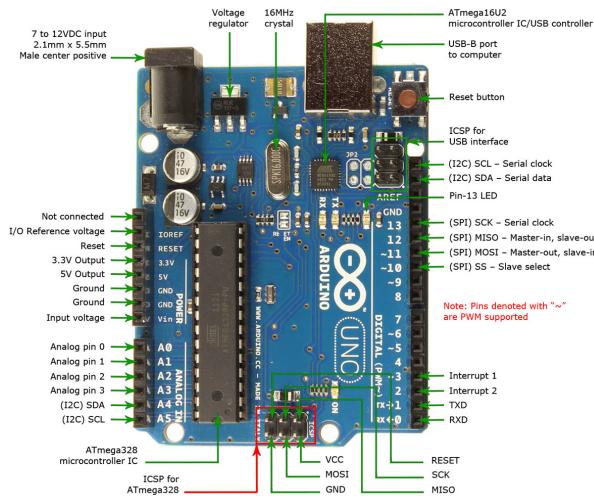


Figura 3.5: Diagrama de conexiones y pines de Arduino UNO rev3.

#### 3.1.3. Sensores

Se han usado diferentes tipos de sensores para medir las principales magnitudes que afectarán al cuidado y a la monitorización de las plantas del invernadero. Estas magnitudes son temperatura y humedad ambiente, que se ha medido tanto en el interior como en el exterior del invernadero, para esto se ha utilizado el módulo DHT22, temperatura de la tierra para la que se ha usado una sonda de temperatura DS18B20 y humedad de la tierra, que se ha medido con un módulo YL-69.

En sus fichas técnicas podemos destacar diferentes características que nos son especialmente relevantes para cada uno de los sensores mencionados anteriormente:

- Sensor DHT22:

- 3-5V de entrada/salida.
- Rango de temperaturas de -40 a 80 °C. Con una precisión de  $\pm 0.5$  °C
- Lectura de humedad en un rango de 0-100 %. Con una precisión de  $\pm 5$  puntos porcentuales.
- Tasa de muestreo de 0.5 Hz.

- Sensor DS18B20:

- 3-5.5V de entrada/salida.

### 3.1. HARDWARE

- Rango de temperaturas de -55 a 125 °C. Con una precisión de  $\pm 0.5$  °C
  - Tasa de muestreo de 1.3 Hz.
- Sensor YL-69:
- 2-6V de entrada/salida.
  - Lectura de humedad analógica. Con una precisión del 5-10 %.
  - Tasa de muestreo de 0,5 Hz.



Figura 3.6: Módulo DHT22.



Figura 3.7: Sonda DS18B20.

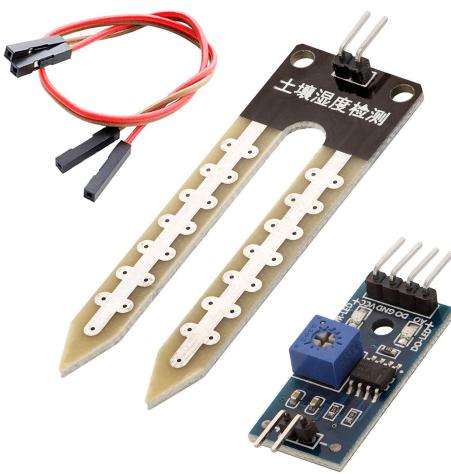


Figura 3.8: Módulo YL-69.

### 3.1. HARDWARE

---

#### 3.1.4. Actuadores

Los actuadores son útiles en cualquier proyecto de IoT, pues utilizando la información obtenida por los sensores, permiten desarrollar sistemas complejos basados en toma de decisiones e interactuar con el medio. En este TFG se ha implementado un sistema de riego automático basado en la humedad de la tierra. Tal y como se indica en la sección de PROYECTOS FUTUROS se podría usar esta información proporcionada por los sensores para incluir nuevas funcionalidades en el sistema y automatizar diferentes tareas.

Para el funcionamiento del sistema de riego se ha usado una bomba de agua modelo CESFONJER, funciona a 12V DC, tiene una potencia de levantamiento de agua de hasta 3 metros y un caudal de 250L/h.

Se ha añadido al sistema electrónico un módulo de 8 relés, cuyo propósito es el de poder controlar la bomba, que funciona a 12V y está conectada a la corriente eléctrica. En la sección de MONTAJE se indica el uso de esta placa.

El tener un módulo de 8 relés proporciona la posibilidad de añadir diferentes actuadores para trabajos futuros como motores, bombas de agua o luces.



Figura 3.9: Bomba de agua modelo CESFONJER.

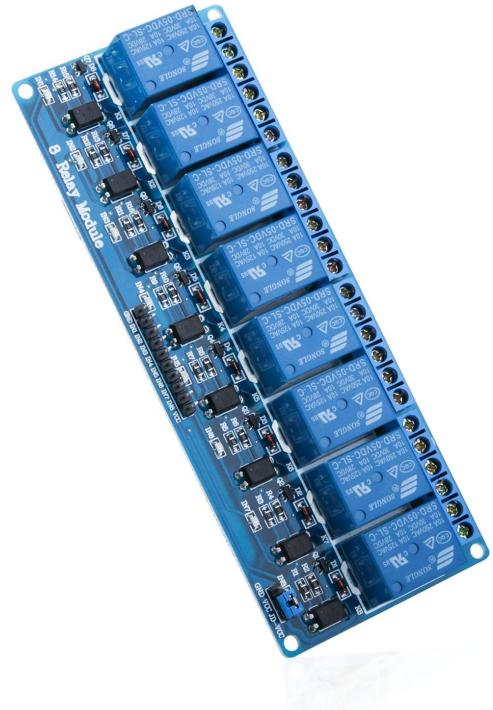


Figura 3.10: Módulo de relés.

## 3.2. Software

En esta sección se presentan las diferentes herramientas software utilizadas para el desarrollo del proyecto. Todas las herramientas utilizadas son gratuitas.

### 3.2.1. Raspbian

Raspberry Pi OS, popularmente conocido como Raspbian, es una distribución de GNU/Linux, y es el sistema operativo típicamente utilizado en cualquier modelo de Raspberry. Desde 2015 se proporciona de forma oficial como el sistema operativo por defecto para estas placas. Se trata de un SO libre y tiene, tal y como cualquier distribución GNU/Linux, multitud de posibilidades. Toda la funcionalidad de la Raspberry para este TFG se ha implementado sobre Raspbian.

### 3.2.2. Python

Python es un lenguaje de programación de alto nivel, interpretado, multiplataforma, de código abierto y de propósito general. Una de sus grandes ventajas es la sencillez y la legibilidad a la hora de desarrollar aplicaciones usando este lenguaje, además, de la multitud de librerías y frameworks disponibles. Soporta no solo la programación orientada a objetos sino también la programación imperativa y la funcional. Fue desarrollado a finales de la década de los 80 por Guido Van Rossum, investigador del centro holandés Centrum Wiskunde & Informatica. Su uso está hoy en día ampliamente extendido. Es muy común en diferentes áreas de la ciencia y la ingeniería y es uno de los lenguajes de programación más solicitados en el mercado. En este TFG se han usado dos versiones diferentes de Python, la Python 3.8 y la 3.5, debido al soporte de las diferentes librerías. Los principales módulos utilizados se irán referenciando a lo largo de la siguiente sección.

Respecto al estilo del código se ha seguido la guía de estilo PEP 8. Estas guías no son de obligado cumplimiento, sin embargo, en general, es de agradecer una unanimidad en dicho estilo. PEP 8 está muy extendida entre la comunidad que desarrolla en Python y ayuda a la legibilidad del código.

#### 3.2.3. Bash

GNU Bash (Bourne-again shell), popularmente conocido como Bash es un intérprete de comandos propio de GNU/Linux. Bash scripting consiste en utilizar Bash como un lenguaje de programación usando las instrucciones propias de este intérprete. Cuando se habla de Bash scripting se habla de un tipo de Shell Scripting propio de los sistemas GNU/Linux, existen muchos otros, no solo para Linux, sino para cualquier sistema operativo como pueden ser Cmd.exe, Csh, AppleScript, COMMAND.COM, ect.

#### 3.2.4. Dash

Dash es un framework de código abierto utilizado para desarrollar aplicaciones de visualización de datos interactivas usando Python, R, Julia o F#. Está escrito sobre Plotly.js y React.js. Las aplicaciones web creadas con Dash son servidores web Flask, un framework escrito en Python para crear aplicaciones web, que se comunican usando JSON, un formato ligerito para intercambiar datos, sobre HTTP (Hypertext Transfer Protocol). El front-end Dash está implementado con React.js, una librería de JavaScript para la interfaz de usuario escrita y mantenida por Facebook. Dash además de las diferentes tecnologías citadas anteriormente tiene embebidos HTML y CSS. En este TFG se ha usado, además, Bootstrap, que se explica a continuación y Plotly, una librería de Python utilizada para crear gráficos interactivos.

#### HTML en Dash

HTML (HyperText Markup Language), es un lenguaje de marcado para la elaboración de páginas web. Actualmente es un estándar en Internet a cargo de W3C (World Wide Web Consortium). El origen de HTML data de principios de la década de los 80, fue creado por Tim-Berners Lee, físico en el CERN (European Organization for Nuclear Research), como un sistema para poder compartir documentos de manera más sencilla a la utilizada hasta el momento. A lo largo de los años se ha ido actualizando e incluyendo multitud de mejoras, actualmente se utiliza HTML5, lanzado en el año 2005. El uso de HTML en Dash se realiza mediante la librería Dash.html, que incluye todos los elementos de HTML, a continuación podemos ver un breve ejemplo de uso extraído de su documentación oficial y de su equivalencia en HTML clásico:

---

```
from dash import html
```

## 3.2. SOFTWARE

```
html.Div([
    html.H1('Hello Dash'),
    html.Div([
        html.P('Dash converts Python classes into HTML'),
        html.P("This conversion happens behind the scenes by Dash's JavaScript front-end")
    ])
])

<div>
    <h1>Hello Dash</h1>
    <div>
        <p>Dash converts Python classes into HTML</p>
        <p>This conversion happens behind the scenes by Dash's JavaScript front-end</p>
    </div>
</div>
```

### CSS en Dash

Las hojas de estilo en cascada, CSS(Cascading Style Sheets), son empleadas para dar estilo a un documento escrito con un lenguaje de marcado como puede ser HTML, fue desarrollado por W3C en 1996. Hoy en día la relación entre CSS y HTML es muy alta, de hecho, CSS es también un estándar en Internet y en el diseño web. En Dash también existe la posibilidad de dar estilo usando CSS, para el desarrollo de esta aplicación se ha utilizado Bootstrap 5.

Figura 3.11:



Figura 3.11: Página de ejemplo sin estilo.

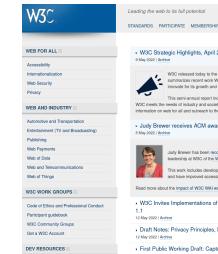


Figura 3.12: Página de ejemplo con estilo CSS.

### Bootstrap en Dash

Este framework de software libre, muy utilizado en programación web se utiliza como una abstracción de HTML, CSS y JavaScript cuyo propósito es facilitar la creación de sitios web para múltiples dispositivos (*responsive design*). Fue desarrollado inicialmente en el año 2011 por ingenieros de la compañía estadounidense Twitter, Inc. Su versión más reciente es Bootstrap 5, lanzada en el año 2021, cuya principal diferencia respecto a sus anteriores versiones es que

### **3.2. SOFTWARE**

---

utiliza Vanilla JavaScript (JavaScript puro), en lugar de JQuery, una librería de JavaScript muy utilizada para agregar interactividad a los sitios web. Como se ha mencionado anteriormente, se ha utilizado una plantilla libre de Bootstrap 5 a la que se le han hecho ligeras modificaciones de estilo, se puede usar este framework en Dash usando la librería Dash\_bootstrap\_components.

#### **3.2.5. API de telegram**

Una API (Application Programming Interfaces) o interfaz de programación de aplicaciones en español, es un conjunto de protocolos y definiciones que se utilizan para desarrollar software y que permite la comunicación entre dos aplicaciones. Telegram es una aplicación de mensajería instantánea desarrollada por los hermanos Nikilái y Pável Dúrov, lanzada el 14 de agosto de 2013. Telegram permite multitud de servicios, entre ellos la posibilidad de utilizar bots. Los bots de telegram tienen una gran popularidad y muy diferentes usos, con ellos se puede desde obtener información hasta realizar descargas o traducir textos.

Telegram ofrece una API para el desarrollo de bots, que es la que se ha utilizado en este TFG junto a la librería Ttelepot para Python.

#### **3.2.6. MongoDB**

MongoDB es un sistema de bases de datos de código abierto, no relacional y orientado a documentos. Utiliza estructuras BSON (Binary JSON) para guardar los datos. Una de las principales ventajas de MongoDB es su velocidad y aprovechamiento de los recursos.

Las bases de datos no relacionales son muy utilizadas en proyectos de IoT y, en general, en proyectos para los que no sea necesario guardar relaciones entre tablas al estilo SQL (Structured Query Language). MongoDB permite acceder a una gran cantidad de datos en poco tiempo, permite realizar multitud de consultas, expresiones regulares, etc.

Es habitual trabajar con Python y MongoDB, para ello se ha utilizado la librería Pymongo, es una de las más extendidas para este propósito y está en continuo desarrollo.

#### **3.2.7. Cron**

Se ha usado Cron para administrar diferentes tareas, Cron es un demonio (administrador regular de procesos en segundo plano) de UNIX. Permite ejecutar procesos y tareas basados en

### 3.2. SOFTWARE

---

tiempo. En este TFG se ha usado Cron para lanzar periódicamente scripts para la realización de diferentes tareas como la recolección automática de los datos de los sensores.

#### 3.2.8. Docker

Docker es un sistema basado en el funcionamiento de virtualización ofrecida por el núcleo de Linux. Este prepara máquinas virtuales de proceso y usa diferentes tecnologías como Cgroups, que permite aislar y limitar los recursos consumidos por un grupo de procesos o el espacio de nombrado de Linux (Linux kernel namespaces), que permite aislar los PID, interfaces de red, tablas de encaminamiento, puntos de montaje, etc. Docker es un sistema muy extendido y de código abierto mantenido por Docker Inc. Algunos de los principales organismos contribuyentes a este proyecto son Google, Cisco, Microsoft, RedHat, entre otras importantes compañías, además de la comunidad. En este caso se ha utilizado Docker para realizar el despliegue de la base de datos de MongoDB en una máquina de los laboratorios de la ETSIT de la Universidad Rey Juan Carlos.

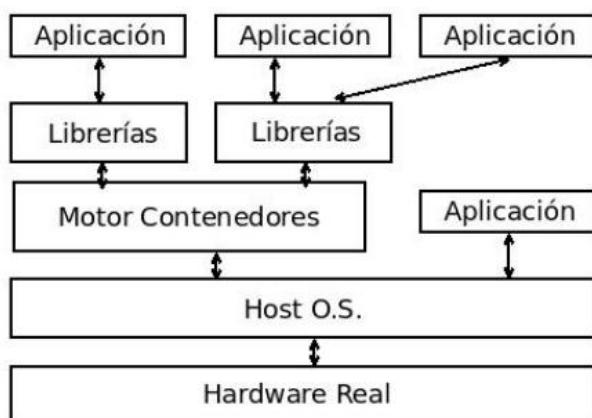


Figura 3.13: Diagrama simplificado del funcionamiento de la virtualización por contenedores.

#### 3.2.9. Heroku

Heroku, fundada en 2007 por Orion Henry, James Lindenbaum y Adam Wiggins y desde 2010 mantenida por Salesforce Inc., es una plataforma de servicios en la nube o PaaS (Platform as a Service), en inglés, que permite manejar la administración y configuración de los servidores. Las principales características de esta popular herramienta son:

### **3.2. SOFTWARE**

---

- Soporta diferentes lenguajes de programación (Ruby, PHP, Python, etc).
- Ofrece diferentes características de seguridad como certificación SSL y autenticación.
- Tiene una versión gratuita de manejo sencillo.

Para este proyecto se ha utilizado su versión más sencilla para desplegar la aplicación web de visualización de datos desarrollada usando Dash. Existen otras alternativas pero gracias a su facilidad de uso ha sido elegida sobre el resto.

#### **3.2.10. Control de versiones**

El control de versiones es esencial en cualquier proyecto de desarrollo de software, se ha utilizado Git y Github para este propósito.

Git es el sistema de control de versiones más extendido actualmente, se trata de un proyecto de software libre en continuo crecimiento. Desarrollado por Linus Torvalds en el año 2005. Git es un sistema de control de versiones distribuido, de tal manera que cada copia del trabajo es también un repositorio que alberga todo el historial de cambios.

Una de las principales ventajas del uso de Git es su rendimiento, y es que gracias a su popularidad, que lo ha convertido prácticamente en un estándar, las bases de sus características son muy sólidas y sus principales funciones están muy optimizadas. Otro aspecto a resaltar de este software es la seguridad, utiliza SHA1, un algoritmo criptográficamente seguro. Todo esto ha llevado a Git a ser el sistema de control de versiones más usado. Github es una plataforma diseñada para alojar código, comprada por Microsoft en el año 2018, permite gestionar proyectos usando Git. Es también muy utilizado y permite alojar proyectos tanto de forma pública como privada.

### 3.2. SOFTWARE

---

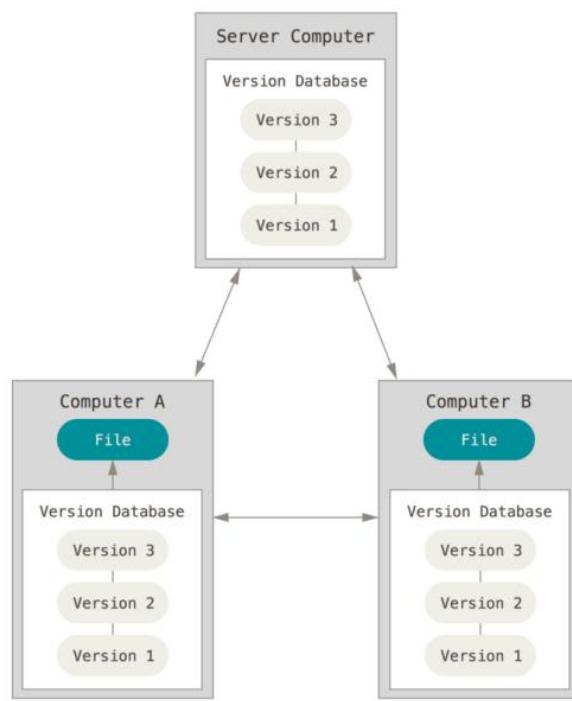


Figura 3.14: Esquema de concepto básico de Git.

# **Capítulo 4**

## **Diseño e implementación**

En este capítulo se expone la implementación del sistema que se ha llevado a cabo en este TFG. Se detalla cada una de las herramientas que se han desarrollado tanto desde el punto vista de hardware como desde el de software.

### **4.1. Montaje físico del invernadero**

En esta sección se explica el montaje físico del invernadero y se detalla su arquitectura. El invernadero ha diseñado para ser de bajo coste. Ha sido fabricado en plástico, mide 2.5x1.25x0.75 metros y en su interior se han introducido sacos de cultivo para plantar. Se ha elegido este tipo invernadero ya que se considera adecuado teniendo en cuenta el espacio disponible para su ubicación. En la figura 4.1 se muestra el invernadero que se ha montado para el desarrollo de este TFG.



Figura 4.1: Implementación física del invernadero.

#### 4.1. MONTAJE FÍSICO DEL INVERNADERO

En este montaje se ha incluido un depósito de agua. El depósito cuenta con una capacidad de 40 litros, capacidad que permite realizar las pruebas del sistema. Otra opción hubiera consistido en utilizar una toma de agua corriente en lugar de un depósito. También se ha de tener en cuenta que el depósito debe llenarse de alguna forma, ya sea a través de una toma de agua o manualmente. En el montaje propuesto en este TFG el depósito está desacoplado de una toma de agua, por lo que debe llenarse manualmente. Aunque existe en este caso la limitación de acceso al agua, nos permite por otro lado que el invernadero no dependa de infraestructuras externas. Otra opción viable y que no implicaría ningún cambio sería, por ejemplo, sustituir el depósito por un pozo. Para poder llevar el agua hasta las plantas se ha usado una tubería de riego con puntos de salida de agua regulables tipo araña conectado a una bomba como se puede observar en las figuras 4.2 y 4.3. La bomba de agua modelo CESFONJER será el actuador que se controlará mediante el software desarrollado para realizar el riego del sistema.



Figura 4.2: Bomba de agua en el depósito.



Figura 4.3: Aquí va una imagen de las salidas de agua.

Se ha utilizado una caja de registro hermética para proteger toda la electrónica como se muestra en las figuras 4.4 y 4.5 . Esta caja está situada dentro del invernadero y contiene el Arduino y la Raspberry Pi, donde se ejecuta el software que se detallará en la sección 4.3 y la protoboard y el módulo de relés detallados en la sección 3.1. De la caja salen los cables que están conectados a los sensores, colocados en su ubicación final.

## 4.2. ARQUITECTURA E IMPLEMENTACIÓN HARDWARE

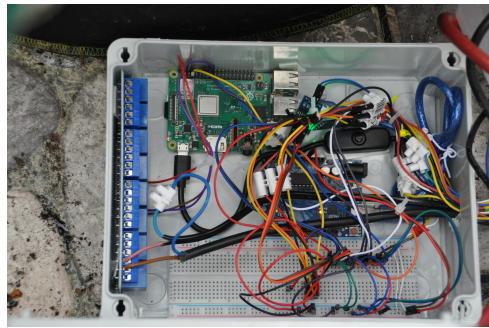


Figura 4.4: Caja de registro abierta.



Figura 4.5: Caja de registro cerrada.

Los sensores que miden humedad y temperatura ambiente se han colocado por parejas de manera que uno de ellos se encuentra fuera del invernadero y su pareja en el interior. Esto permite realizar comparativas entre la temperatura y la humedad externa y la interna. Los sensores de humedad y temperatura de la tierra se han colocado cercanos entre sí para poder realizar las pruebas del sistema, como por ejemplo poder monitorizar una sola zona de la tierra y comprobar la diferencia entre sus medidas o comprobar que estos funcionan correctamente.

## 4.2. Arquitectura e implementación hardware

El montaje de la electrónica que se presenta en este apartado incluye, como se especifica en el apartado 3.1, un microcomputador Raspberry Pi Model 3B+, conectado por su puerto USB a una placa Arduino y varios sensores y actuadores conectados a ambos dispositivos.

La Raspberry Pi está conectada al Arduino usando los puertos USB de ambos dispositivos.

La placa Arduino cuenta con pines analógicos, que no se encuentran en la Raspberry Pi. Estos pines permiten utilizar sensores analógicos que abaratan significativamente los costes del proyecto. Un ejemplo de este tipo de sensor son los higrómetros que han sido utilizados en este TFG. Una solución alternativa a este problema sería utilizar un conversor analógico-digital, aunque el coste total ascendería. La elección de la primera opción se debe principalmente a dos motivos, primero, como se ha mencionado anteriormente, es más barato un Arduino que un conversor de este tipo y segundo, nos proporciona una mayor cantidad de pines analógicos, digitales y de salida de voltaje para poder extender el modelo implementado o añadir nuevos sensores en proyectos futuros.

A continuación se detalla el montaje de cada uno de los sensores explicados en la sec-

## 4.2. ARQUITECTURA E IMPLEMENTACIÓN HARDWARE

---

ción 3.1.3. Cada sensor cuenta con tres entradas diferentes la entrada de datos, la entrada de voltaje (Vcc), la tierra (GND). La entrada de datos ha sido conectada a las placas como se muestra a continuación, las entradas de voltaje y tierra a una protoboard.

- El primer módulo con sensor DHT22 al GPIO 17 de la Raspberry.
- El segundo módulo con sensor DHT22 al GPIO 27 de la Raspberry.
- La primera sonda DS18B20 al GPIO 04 de la Raspberry.
- La segunda sonda DS18B20 al GPIO 04 de la Raspberry.
- El primer módulo YL-69 al pin A0 del Arduino.
- El segundo módulo YL-69 al pin A1 del Arduino.

Con respecto a los sensores DS18B20 cabe destacar que ofrecen la posibilidad de realizar lecturas en serie de varios dispositivos. Este tipo de sensores utiliza el protocolo *I-Wire* y permite conectar más de un sensor en el mismo bus para realizar la comunicación. Esto permite utilizar menos entradas, lo que facilita la inclusión de nuevos sensores en el futuro al disponer de más pines para conectarlos.

Tal y como se ha comentado anteriormente, además de la entrada de datos es fundamental conectar cada sensor y actuador a una fuente de alimentación y a tierra. Para facilitar la gestión del cableado, se ha utilizado una protoboard. La protoboard ha sido alimentada con el pin de 5V y GND del Arduino. Todas las conexiones electrónicas se han realizado usando cables Dupont, macho-hembra, macho-macho o hembra-hembra según procediese.

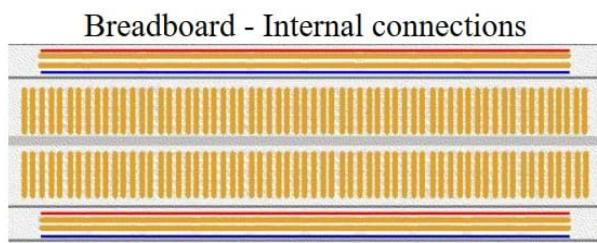


Figura 4.6: Conexiones internas Protoboard.

El conexionado de todos los sensores es equivalente. En la figura ?? se muestra un esquema del conexionado para un sensor.

## 4.2. ARQUITECTURA E IMPLEMENTACIÓN HARDWARE

---

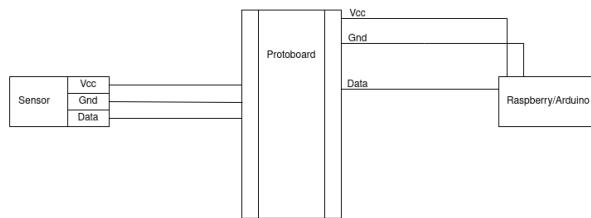


Figura 4.7: Conexiones internas Protoboard.

Como se indica en el apartado 3.1.4, para controlar los actuadores desde el software desarrollado en este TFG se ha añadido un módulo de relés, conectado a la Raspberry Pi. La bomba de agua funciona a la corriente eléctrica doméstica (240 V), por lo que no se puede alimentar directamente con el voltaje proporcionado por la Raspberry o el Arduino. Para poder apagar o encender los actuadores, se ha colocado el relé en serie a modo de interruptor entre dos de los bornes del actuador, que estará conectado a la corriente eléctrica. El diagrama de la figura 4.8 muestra como se ha diseñado dicho circuito.

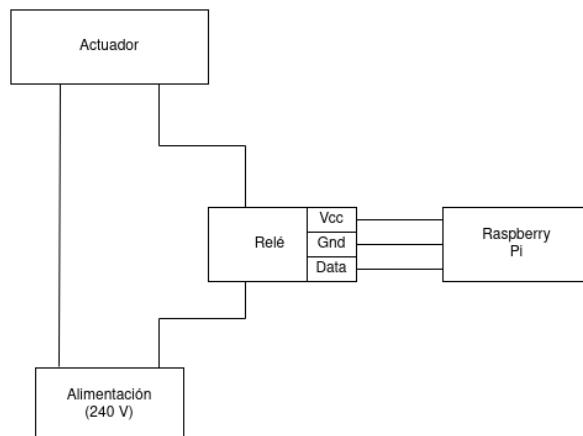


Figura 4.8: Esquema de conexionado para un actuador.

A su vez, cada uno de los actuadores está asociado a un pin GPIO de la Raspberry, en concreto, la bomba CESFONJER está asociada al GPIO 24. Físicamente esta conexión se realiza mediante tres cables dupont que conectan el módulo de relés con la Raspberry Pi, para las entradas de Vcc, tierra y datos.

## 4.3. Arquitectura e implementación software

### 4.3.1. Arquitectura general

El código desarrollado en este TFG está disponible en Github<sup>1</sup>. El repositorio se divide en *code/*, donde se puede encontrar todo el código desarrollado, y en *doc/*, con la documentación sobre el mismo. En *code/* podemos encontrar otras cinco subcarpetas:

- *apps*, código para el desarrollo de las aplicaciones que se han realizado.
- *backup*, código para realizar un backup de la base de datos.
- *controllers*, código relativo a los actuadores.
- *datacollecting*, código relativo a la recolección de datos y los sensores.
- *modules*, credenciales de acceso a la base de datos.
- *test*, test para comprobar el correcto funcionamiento de los sensores.

En la figura 4.9 se muestra la arquitectura del software implementado en este TFG. Se han realizado dos aplicaciones relativas a la visualización de datos y monitorización del sistema, una aplicación web para la visualización de datos con Dash, que se ha desplegado en Heroku<sup>2</sup>, y un bot de Telegram<sup>3</sup>, que está desplegado en la propia Raspberry, y que permite además recibir notificaciones sobre el estado del invernadero, las medidas tomadas y los actuadores. Existe también, una aplicación para controlar el riego en el invernadero mediante los actuadores del sistema.

---

<sup>1</sup><https://github.com/rinvictor/TFG>

<sup>2</sup><https://greenhouse-tfg.herokuapp.com>

<sup>3</sup>[t.me/greenhouseTFGBot](https://t.me/greenhouseTFGBot)

#### 4.3. ARQUITECTURA E IMPLEMENTACIÓN SOFTWARE

---

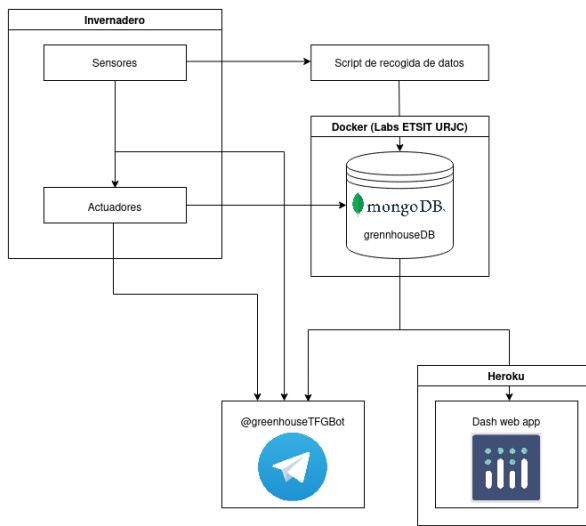


Figura 4.9: Esquema de la implementación software del sistema.

La figura 4.10 muestra la base de datos que se ha utilizado. La base de datos se ha desplegado en los laboratorios docentes de la ETSIT de la URJC en un contenedor Docker, en concreto en la máquina *l2018-pc05.aulas.etsit.urjc.es*. Esta base de datos consta de dos colecciones, *sensors\_data* y *controllers\_data*. En *sensors\_data* se almacenan los datos relacionados con la información obtenida por los sensores mientras que en *controllers\_data* se almacenan la información relacionada con las acciones que toman los actuadores.

En la figura 4.11 se muestra la arquitectura que se ha elegido para guardar la información de los sensores. Por cada lectura se almacena el ID del documento, el ID del sensor, el tipo de dato que se almacena, es decir, la magnitud medida, el valor leído, la fecha de lectura y la localización del sensor.

En la figura 4.12 se puede observar la arquitectura para la información sobre las acciones tomadas por los actuadores. En cada documento se almacena el ID del documento, el ID del controlador, la fecha de la acción realizada y el tipo de acción, por ejemplo, riego.

Se almacenan seis valores por cada lectura del sensor. Se realizan un total de ocho lecturas, (dos por sensor), cada dos horas en *sensors\_data*. Para el caso de *controllers\_data* se almacena un valor una vez al día.

#### 4.3. ARQUITECTURA E IMPLEMENTACIÓN SOFTWARE

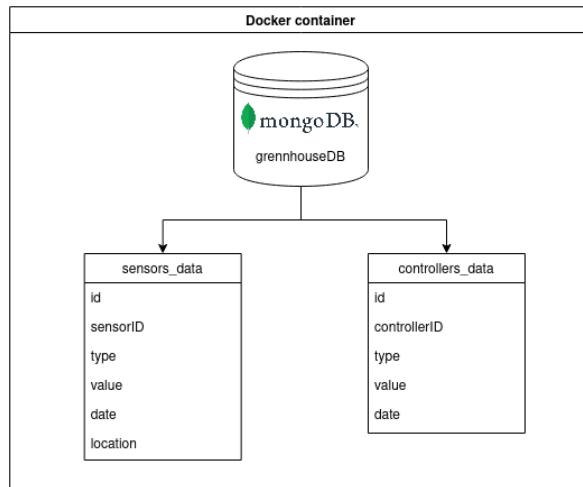


Figura 4.10: Esquema general de la estructura de la base de datos.

▼	(1) ObjectId("6202b99a9d23b38270ebeb9a")	{ 6 fields }	Object
└─	_id	ObjectId("6202b99a9d23b38270ebeb9a")	ObjectId
└─	date	2022-02-08 19:42:34.503Z	Date
└─	type	ground humidity	String
└─	location	indoor	String
└─	value	0.998	Double
└─	sensorID	0	Int32

Figura 4.11: Estructura de la colección *sensors\_data*, perteneciente a la base de datos *greenhouseDB*.

▼	(1) ObjectId("6291103fedefecf01f062eb3")	{ 5 fields }	Object
└─	_id	ObjectId("6291103fedefecf01f062eb3")	ObjectId
└─	value	10.7	Double
└─	date	2022-05-27 19:54:07.872Z	Date
└─	type	irrigation	String
└─	controllerID	24	Int32

Figura 4.12: Estructura de la colección *controllers\_data*, perteneciente a la base de datos *greenhouseDB*.

Se realiza una copia de seguridad de cada una de las colecciones de la base de datos una vez al día, dejándola en dos archivos .csv. De esta manera el sistema puede sobreponerse al borrado no intencionado de la información como por ejemplo, si el contenedor es borrado. El código para realizar este backup se localiza en *TFG/code/backup*. La lógica del proceso se encuentra en *backup.py* mientras que el proceso que lanza la lógica se encuentra en *execute\_backup.sh*. Al ejecutar el proceso anterior se almacenan los mensajes en *TFG/code/backup/logs/success.log* si todo funciona correctamente, en *TFG/code/backup/logs/error.log* si falla y en *TFG/code/backup/logs/cronerror.log* si es cron el que encuentra problemas para ejecutar el proceso.

## **4.3. ARQUITECTURA E IMPLEMENTACIÓN SOFTWARE**

---

El software para la de recogida de datos, control de actuadores y el bot de Telegram se ha desplegado en la Raspberry Pi, mientras que la aplicación web se ha desplegado en la nube y la base de datos está alojada en los laboratorios docentes de la URJC. De esta forma, si hubiera una rotura o una caída de la red en la Raspberry podría detectarse rápidamente a través de la aplicación web, y los datos almacenados hasta el momento no se perderían y podrían seguir siendo consultados. Por otro lado, la caída de la aplicación web no afectaría al funcionamiento de la Raspberry Pi y todos los procesos ejecutados en ella continuarían. Por último, si la base de datos dejase de funcionar se podría detectar rápidamente a través de los logs de la Raspberry y mediante el bot de Telegram.

Los scripts de recogida de datos, creación del backup de la base de datos, comprobación de la calidad de los datos y la toma de acciones se lanzan en segundo plano utilizando Cron. A continuación se muestra el fichero de configuración para lanzar dichos scripts.

---

```
1 */2 * * * /home/pi/TFG/code/datacollecting/execute.sh > /home/pi/TFG/code/datacollecting/logs/cronerror.log 2>&1
10 */2 * * * /home/pi/TFG/code/apps/telegram/execute_monitoring.sh > /home/pi/TFG/code/apps/telegram/logs/cronerror.log 2>&1
1 *24 * * * /home/pi/TFG/code/backup/execute_backup.sh > /home/pi/TFG/code/backup/logs/cronerror.log 2>&1
30 *24 * * * /home/pi/TFG/code/apps/actions/execute_actions.sh > /home/pi/TFG/code/apps/actions/logs/cronerror.log 2>&1
```

---

Se pueden observar los diferentes *.sh* que se han desarrollado para ejecutar las aplicaciones y como, en caso de fallo, se escribe el fichero *cronerror.log*. En *execute.sh* se lanza el software para la recogida de datos, *execute\_monitoring.sh* es el encargado de lanzar el software para comprobar la calidad de los datos y el correcto funcionamiento de los sensores, *execute\_backup.sh* lanza el backup de la base de datos y *execute\_actions.sh* lanza el software para controlar los actuadores y realizar acciones. La funcionalidad de estos scripts es comentada en las siguientes secciones.

### **4.3.2. Recogida de datos**

La recogida de datos, como se ha mencionado anteriormente, se realiza en la propia Raspberry. Para implementar esta funcionalidad se han creado varios ficheros y el código está disponible en *code/datacollecting*. En primer lugar, se ha creado el fichero de configuración *config.py* en el cual se implementa la clase Sensor, que facilita la gestión de la localización de los sensores y de los pines a los que están conectados.

---

```
class Sensor:
    def __init__(self, pin, location):
        self.pin = pin
```

#### **4.3. ARQUITECTURA E IMPLEMENTACIÓN SOFTWARE**

---

```
self.location = location
```

---

Todos los sensores en este TFG se expresan en el código con instancias de la clase *Sensor*. El pin al que está conectado se utiliza a modo de identificador único ya que varios sensores no pueden estar conectados al mismo pin. El caso de los sensores DS18B20 es diferente, ya que como se explica en el apartado 4.2 estos sensores si pueden estar conectados a un mismo pin, por lo que se utiliza su identificador hardware como identificador único. La localización indica si el sensor se encuentra dentro o fuera del invernadero.

Los scripts para la recogida de datos se han implementado en los ficheros *sensors.py* y *db\_conn.py*. En *sensors.py* se definen las diferentes funciones que permiten leer la información de los sensores, habiendo implementado una función para cada tipo de sensor. Para este propósito se han utilizado distintas librerías que permiten controlar los sensores: *pyfirmata2* para los sensores YL-69, *Adafruit\_DHT* para los sensores DHT22 y *wlthermsensor* para los DS18B20. Existen las funciones *get* y las funciones *read*. Las primeras sirven para manejar los datos recogidos por las funciones *read* de tal manera que las funciones *read* se encargan de leer los datos mediante los sensores y las funciones *get* de obtenerlos para manejarlos desde distintos puntos en el software.

En el fichero *db\_conn.py* se implementa la funcionalidad relativa a la inserción de los datos en la base de datos *greenhouseDB* de MongoDB, en concreto en la colección *sensors\_data*. Antes de insertar la información en la base de datos, se valida para detectar inconsistencias. En el caso de que la lectura sea nula (en Python *None*) o que sea un valor fuera de un rango establecido (por ejemplo, 120 °C de temperatura es un error) se almacena un -1. Puede interesar conocer, por ejemplo, la tasa de fallo de un sensor, es por este motivo que no se hacen reintentos en la lectura. Un fallo en una lectura queda escrito en los ficheros de log, además, como se comenta posteriormente, se le comunica al usuario vía Telegram. A continuación se muestra un pequeño fragmento a través del cual se inserta el valor de la temperatura ambiente medida por los sensores DHT22 en la base de datos. Como se puede observar, se almacena el tipo de dato, el valor obtenido, el ID del sensor, la localización y la fecha de captura del dato.

---

```
now = datetime.now()

item = {"type": "ambient temperature", "value": ambient_temperature, "sensorID": d.pin, "location": d.location,
        "date": now}
collection.insert_one(item)
```

---

## 4.3. ARQUITECTURA E IMPLEMENTACIÓN SOFTWARE

---

Se ha desarrollado un script de Bash donde se lanzan los scripts de Python encargados de la recolección de datos, además se realizan las exportaciones de módulos de Python pertinentes y se almacena el resultado del proceso en los ficheros de log. El script de Bash si realiza reintentos, de tal manera que si es la aplicación la que falla relanza su ejecución. En caso de fallo se almacena el resultado con la etiqueta *Exec Error*. En caso de fallar este segundo reinicio la etiqueta pasa a ser *Fatal Error*. Si el segundo intento tiene éxito, se escribe en el fichero de log indicando que ha sido tras un reinicio. Todos los ficheros de log se escriben con la fecha y el resultado de la ejecución. A continuación se muestra un ejemplo de la información que se almacena en los *logs*.

---

```
vie may 27 20:01:27 CEST 2022 success {'type': 'ground humidity', 'value': 0.2278, 'sensorID': 0, 'location': 'indoor', 'date': datetime.datetime(2022, 5, 27, 20, 1, 8, 24826), '_id': ObjectId('629111e4b2c6d1b9a5e7f23f')} 0.2209 {'type': 'ground humidity', 'value': 0.2209, 'sensorID': 1, 'location': 'indoor', 'date': datetime.datetime(2022, 5, 27, 20, 1, 13, 710340), '_id': ObjectId('629111e9b2c6d1b9a5e7f240')} 32.0 {'type': 'ambient humidity', 'value': 32.0, 'sensorID': 22, 'location': 'outdoor', 'date': datetime.datetime(2022, 5, 27, 20, 1, 15, 457495), '_id': ObjectId('629111ebb2c6d1b9a5e7f241')} 27.200000762939453 {'type': 'ambient humidity', 'value': 27.200000762939453, 'sensorID': 27, 'location': 'indoor', 'date': datetime.datetime(2022, 5, 27, 20, 1, 17, 203246), '_id': ObjectId('629111edb2c6d1b9a5e7f242')} 28.899999618530273 {'type': 'ambient temperature', 'value': 28.899999618530273, 'sensorID': 22, 'location': 'outdoor', 'date': datetime.datetime(2022, 5, 27, 20, 1, 21, 482673), '_id': ObjectId('629111f1b2c6d1b9a5e7f243')} 32.79999923706055 {'type': 'ambient temperature', 'value': 32.79999923706055, 'sensorID': 27, 'location': 'indoor', 'date': datetime.datetime(2022, 5, 27, 20, 1, 23, 239564), '_id': ObjectId('629111f3b2c6d1b9a5e7f244')} 33.0 {'type': 'ground temperature', 'value': 33.0, 'sensorID': '325a0f1e64ff', 'location': 'indoor', 'date': datetime.datetime(2022, 5, 27, 20, 1, 25, 179596), '_id': ObjectId('629111f5b2c6d1b9a5e7f245')} 27.625 {'type': 'ground temperature', 'value': 27.625, 'sensorID': '32360c1e64ff', 'location': 'indoor', 'date': datetime.datetime(2022, 5, 27, 20, 1, 27, 99572), '_id': ObjectId('629111f7b2c6d1b9a5e7f246')}
```

---

Este tipo de información es útil para poder encontrar fallos en el sistema. Además de los logs resultantes del lanzamiento de los scripts desarrollados en este TFG, también se guardan los errores propios de Cron en el fichero *cronerror.log*.

### 4.3.3. Aplicación de Dash

En este TFG se ha desarrollado una aplicación que permite visualizar de forma gráfica la información recogida por los sensores y las acciones tomadas por los actuadores. Esta aplicación se ha implementado usando el framework Dash, descrito en la sección 3.2.4. Todo el código que usa la aplicación se encuentra dentro de */code/apps/dashapp*. Para el desarrollo de la aplicación se han utilizado los componentes html y CSS embebidos en Dash además de una plantilla libre de Bootstrap para componer el estilo de la página.

La aplicación tiene varios componentes, *app.py*, programa principal, *dash\_utils.py*, donde se han implementado funciones de filtrado y *create\_figures.py*, donde se implementa la generación

#### **4.3. ARQUITECTURA E IMPLEMENTACIÓN SOFTWARE**

de las visualizaciones utilizando la librería *Plotly*.

Una vez se lanza la aplicación y se accede a ella a través del navegador, se encuentran las siguientes funcionalidades, tal y como se muestra en la figura 4.13

- *HOME*, página principal.
- *SENSORS DATA*, un desplegable con las magnitudes medidas.
- *CONTROLLERS DATA*, un desplegable con los controladores (la aplicación de riego).
- *DOCUMENTATION*, documentación del proyecto.

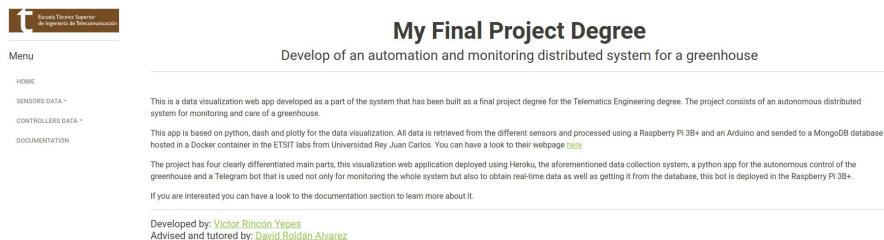


Figura 4.13: Aplicación de visualización de datos.

A través de cada opción del menú se accede a la gráfica que representa la información recogida por los sensores o actuadores de un mismo tipo, por ejemplo, en el desplegable *SENSORS DATA* se pueden ver las distintas magnitudes que se miden y para las que se obtienen datos del sistema, humedad ambiente, temperatura ambiente, humedad de la tierra y temperatura de la tierra. A modo de ejemplo, en la figura 4.14 se puede observar la gráfica que representa la información recogida por el sensores 22 y 27.

#### 4.3. ARQUITECTURA E IMPLEMENTACIÓN SOFTWARE

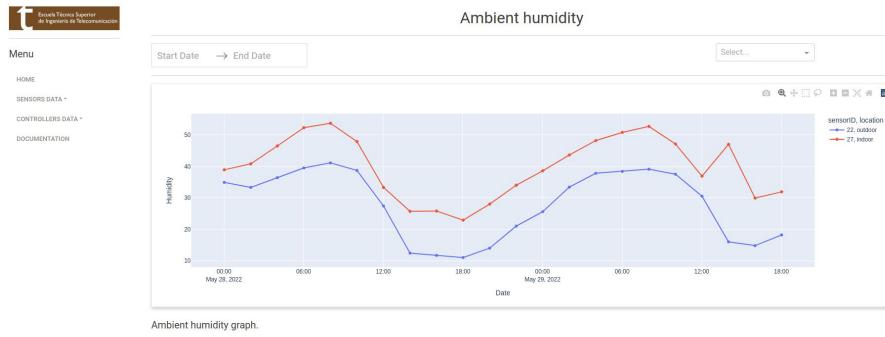


Figura 4.14: Aplicación de visualización de datos.

Entre el título de la figura y la gráfica que se observan en la figura 4.14, se muestran que hay una serie de campos que permiten realizar filtrados en la información presentada. Como ejemplo, en la figura 4.15 se ha seleccionado el sensor 27 para mostrar únicamente la información recogida por dicho sensor.

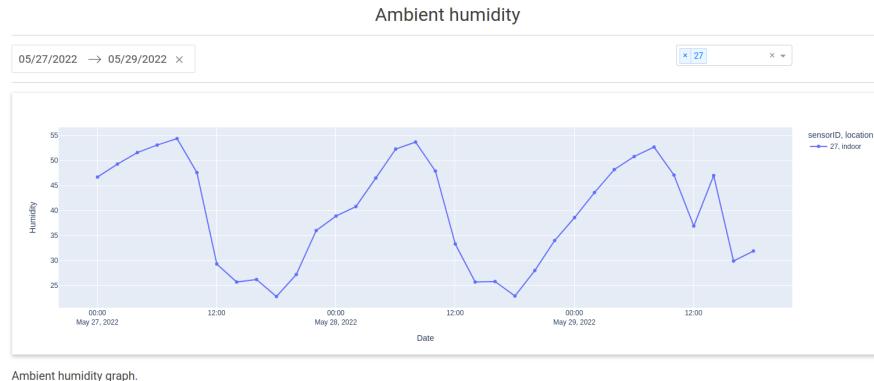


Figura 4.15: Cambio tras selección de fecha y sensor en la humedad ambiente.

La interactividad de la aplicación, que en este caso se consigue a través de los filtros que se han presentado en el párrafo anterior, se consigue con los *callbacks* de Dash. Un *callback* es una función que se lanza automáticamente cada vez que un componente recibe un cambio, pudiendo generar este cambio una actualización de la visualización.

El siguiente código muestra como se implementa una función *callback*. En esta función, se reciben como parámetros de entrada los filtros a través de los que se establecen una fecha de

#### **4.3. ARQUITECTURA E IMPLEMENTACIÓN SOFTWARE**

---

inicio y de fin de los datos, así como el ID del sensor que se quiere visualizar. En base a los filtros, se seleccionan los datos pertinentes y se genera una nueva visualización.

```
@app.callback(
    Output('ambient_temperature_fig', 'figure'),
    [Input('date_picker_ambient_temperature', 'start_date'),
     Input('date_picker_ambient_temperature', 'end_date'),
     Input('dropdown_ambient_temperature', 'value')]
)

def update_output(start_date, end_date, value):
    df = get_dataframe("ambient temperature")
    if not value and not start_date and not end_date:
        # By default values
        yesterday = datetime.strftime(datetime.now() - timedelta(1), '%Y-%m-%d')
        today = datetime.strftime(datetime.now(), '%Y-%m-%d')
        df = du.date_filter(df, start_date=yesterday, end_date=today)
    if start_date or end_date:
        df = du.date_filter(df, start_date, end_date)
    if value and len(value) > 0:
        df = df[df['sensorID'].isin(value)]

    fig = figures.create_temperature_fig(df)
    return fig
```

---

La gestión de la información dentro de la aplicación se realiza utilizando el módulo Pandas de Python. A través de este módulo resulta cómodo y sencillo interconectar la información almacenada en la base de datos MongoDB con la creación de las gráficas correspondientes. Es muy común trabajar utilizando Pandas y MongoDB.

A través de la aplicación Dash se hace una llamada a la base de datos, se guarda la información en un DataFrame de Pandas, que consiste en una tabla con columnas y filas. A este DataFrame se le pueden aplicar filtros para obtener solo una parte de la información y finalmente se utiliza para crear, junto con Plotly, la gráfica que se solicite.

Esta misma funcionalidad es equivalente para mostrar los datos relativos a los actuadores, en concreto, se muestran los litros utilizados al día por la aplicación de riego, cuyo funcionamiento concreto se mostrará en la sección 4.3.5.

Entre las funcionalidades ofrecidas por Plotly, cabe destacar que es posible descargar las imágenes directamente y hacer zoom sobre una zona en concreto de la gráfica.

##### **4.3.4. Bot de Telegram**

En este TFG se ha desarrollado un bot de Telegram llamado *greenhouseTFGBot*. El objetivo de éste bot no es exclusivamente el de enviar mensajes al usuario sino, además, poder obtener información de los sensores, tanto en tiempo real como desde la base de datos y enviar gráficas

#### **4.3. ARQUITECTURA E IMPLEMENTACIÓN SOFTWARE**

---

construidas con Plotly interactuando con el usuario. En esta sección se especifican todas las funcionalidades implementadas en este bot. Ejemplos concretos y salidas de comandos pueden ser vistos en la sección 5.2.

Este bot se ha desarrollando haciendo uso de la API de telegram y de la librería de Python Telepot. Se puede encontrar el código en *code/apps/telegram* del repositorio de Github.

Para crear el bot se ha utilizado un bot propio de Telegram, llamado *Botfather*, que permite crear y administrar los bots ya existentes. Al crear un bot se crea un token de identificación única que será el utilizado para poder desarrollar toda la funcionalidad y asociarla a este bot.

La aplicación principal se encuentra en el fichero *app.py*. Como se ha comentado anteriormente, hace uso de la librería Telepot. Telepot tiene una función incluida llamada *MessageLoop*. Esta función recibe dos parámetros, el objeto que hace referencia al bot, y la función *handle* que manejará los mensajes que el usuario introduzca por pantalla. *MessageLoop* es un bucle infinito que está en continua escucha y responde a las diferentes peticiones del usuario.

Las funcionalidades que se han incluido en este bot son:

- Obtener una lista con todos los comandos disponibles
- Obtener una lista con información sobre cada uno de los sensores
- Obtener una lista con información sobre cada uno de los actuadores
- Obtener el estado de uno o todos los sensores en tiempo real
- Obtener el estado de uno o todos los sensores y actuadores leyendo desde la base de datos.
- Obtener las figuras construidas con Plotly de uno o todos los sensores y actuadores.
- Enviar mensajes de error cuando alguna de las lecturas de cualquier sensor falle.
- Enviar mensajes informativos sobre la ejecución de los actuadores.

Cuando el bot es lanzado con el comando */start*, el usuario recibe un mensaje con los comandos disponibles. El usuario puede volver a pedir esta información con el comando */help*. Cuando se introduce un comando erróneo o inexistente se recibe un mensaje indicando que el comando introducido es incorrecto concatenado con la lista de comandos disponibles.

#### 4.3. ARQUITECTURA E IMPLEMENTACIÓN SOFTWARE

---

Escribiendo `/sensorsinfo` se recibe una lista con información sobre cada uno de los sensores que están implementados en el sistema, indicando su id (GPIO, pin Arduino o su id de hardware). La información sobre los actuadores se recibe usando el comando `/controllersinfo`.

Para obtener información en tiempo real de las medidas tomadas por los sensores se hace uso de los comandos `/sensorsdata`, `/groundhumidity`, `/groundtemperature`, `/ambienthumidity` y `/ambienttemperature`. Estas llamadas devuelven dos mensajes, el primero en el que se indica que se está obteniendo la información solicitada y que va a tardar unos segundos en completarse la acción, y el segundo que devuelve la información solicitada, el id del sensor, su localización y el valor medido. En el caso de hacer `/sensorsdata` devuelve esta misma información pero para todos los sensores del sistema en un mismo mensaje.

Esta funcionalidad hace uso del código desarrollado en `sensors.py` y en `config.py`. En `sensors.py` existen varias funciones que pueden devolver la información solicitada y con un formato concreto.

Como ejemplo, la función que se muestra a continuación recibe un parámetro por defecto, en el caso de ser necesario devuelve el valor leído, en caso de no ser necesario no realiza esta lectura. En la función se puede obtener una lista con la información del sensor y el valor medido por este en el caso de que el parámetro `value` valga `True`. En caso contrario, se obtendría la lista sin el valor medido. Esto permite ahorrar tiempo de ejecución cuando, por ejemplo, solo es necesario obtener un diccionario con los valores `type`, `sensor` y `location`, como ocurre cuando únicamente se necesita obtener información general de los sensores (`/sensorsinfo` y `/controllersinfo`). El fichero `config.py` es el fichero donde se instancian cada uno de los sensores, como se ha indicado anteriormente.

---

```
def get_ground_temp_list(value=True):
    ground_temperature_list = []
    for i in config.ds18b20_list:
        if value:
            d = dict(type="ground_temp", sensor=i.pin, location=i.location, value=read_ground_temperature(i.pin))
        else:
            d = dict(type="ground_temp", sensor=i.pin, location=i.location)
        ground_temperature_list.append(d)
    return ground_temperature_list
```

---

El bot desarrollado también permite obtener la última lectura de los sensores almacenada en la base de datos. De esta manera, no se lanza una petición para recoger información en tiempo real. Estas llamadas son: `/sensorsdatadb`, `/groundhumiditydb`, `/groundtemperaturedb`, `/ambienthumiditydb`, `/ambienttemperaturedb` e `/irrigationdb`. El mensaje devuelto por estos comandos

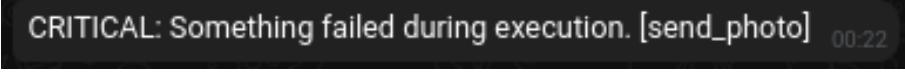
#### **4.3. ARQUITECTURA E IMPLEMENTACIÓN SOFTWARE**

---

es similar a los mencionados anteriormente, con la diferencia de que añade la fecha y hora de la lectura que se está mostrando. De igual manera, `/sensorsdata` devuelve la última lectura de todos los sensores del sistema.

A través de los comandos disponibles en el Bot también se pueden obtener gráficas similares a las presentadas en Dash. Por defecto, devuelve la figura de las mediciones realizadas el mismo día y el día anterior. Para permitir el filtrado de la información es posible añadir parámetros extra al comando. De esta manera es posible indicar una fecha de inicio y de fin para obtener únicamente la información recogida entre dos fechas. En el caso de únicamente incluir la fecha de fin, se devuelve la información almacenada desde la primera lectura hasta la fecha de fin indicada. El formato que esperan estos parámetros es `yyyy-mm-dd`, en el caso de no seguir este formato, se devuelve un mensaje de error. Existen otros mensajes para indicar que el usuario está haciendo un mal uso del comando, como cuando, por ejemplo, se solicita una fecha posterior a la del día en que se realiza la petición o si se escribe mal un parámetro. Las gráficas se obtienen haciendo uso de los comandos `/getallfigures`, `/groundhumidityfig`, `/groundtemperaturefig`, `/ambienthumidityfig`, `/ambienttemperaturefig` e `/irrigationfig`, pudiendo añadir los parámetros opcionales `-startdate` y `-enddate`. Cualquiera de estos comandos devuelve tres mensajes, el primero indica que se está construyendo la figura, el segundo mensaje incluye la propia figura y el tercero indica la localización de los sensores a los que se está haciendo referencia. En el caso de `/getallfigures`, se devuelven todas las figuras, además de un mensaje indicando que el proceso ha terminado.

Esta funcionalidad se ha realizado haciendo uso las funciones desarrolladas en el script `TFG/code/apps/dashapp/dash_utils.py` y `TFG/code/apps/dashapp/create_figures.py`. Las imágenes son enviadas en formato png, convirtiéndolas previamente a bytes. Si existiera algún error durante el procesamiento del comando se enviaría un mensaje de error al usuario como se muestra en la figura 4.16.



CRITICAL: Something failed during execution. [send\_photo] 00:22

Figura 4.16: Mensaje de error enviado cuando el proceso de obtención de la imagen o el envío de esta falla.

Existe, además, el archivo `monitoring.py`, también relativo a este bot de Telegram. Este script

## **4.3. ARQUITECTURA E IMPLEMENTACIÓN SOFTWARE**

---

se ejecuta en cron tras cada lectura de *datacollecting*. Pretende asegurar que los datos recogidos son coherentes, en caso contrario, envía un mensaje al chat del usuario con el bot indicando el posible error. En *monitoring.py*, solo se contempla que el valor obtenido sea -1, ya que la comprobación de persistencia de datos, se realiza a la hora de la recogida. Aunque es posible ignorar los datos erróneos, se ha optado por no hacerlo para poder encontrar facilmente un fallo recurrente de un sensor e incluso poder sacar estadísticas de fallo de dispositivos como se ha especificado en el apartado 4.3.2.

Como se ha comentado anteriormente, el usuario también recibe mensajes automáticos mediante el bot con información de los actuadores, en concreto, recibe el número de litros usado en la aplicación de riego. Esto se realiza desde la propia aplicación como se comenta posteriormente.

### **4.3.5. Aplicación de riego**

En *code/apps/actions* se encuentra el código relativo a los actuadores. En *actions.py* se ha desarrollado una aplicación que sirve para automatizar el riego del sistema. Utiliza el código implementado para la lectura de datos de los sensores para obtener la humedad actual de la tierra. Experimentalmente se ha comprobado que los valores ideales para esta deben estar en torno a 0.2-0.45, esto haría referencia a un 80-55 % de humedad, aproximadamente. Según diversos estudios[?] el porcentaje ideal de humedad de la tierra para el correcto desarrollo de los cultivos se encuentra entre el 50 y el 80 %. Si los valores se saliesen de este rango, podría tener consecuencias negativas para el rendimiento de los cultivos.

Este script se ejecuta en Cron, realiza la medida de la humedad de la tierra y en función del valor obtenido realiza el riego con mayor o menor cantidad de agua. Si el porcentaje es demasiado bajo utilizará más cantidad y si es cercano a los valores ideales utilizará menos. Esta acción se realiza calculando el tiempo necesario que debe estar encendida la bomba. De nuevo la cantidad de agua que se utiliza se ha obtenido de manera experimental. En general, se ha conseguido mantener la humedad dentro de los parámetros deseados, y el gasto de agua se ha visto reducido en comparación con un riego al uso o con un sistema de riego no inteligente. Una vez ha realizado el riego satisfactoriamente se envía un mensaje al usuario por Telegram indicando cuantos litros han sido necesarios para realizarlo. Se almacena este dato, además, en la base de datos, en la colección llamada *controllers\_data*.

#### 4.3. ARQUITECTURA E IMPLEMENTACIÓN SOFTWARE

---

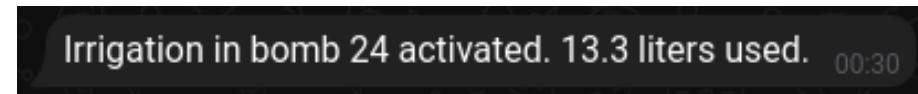


Figura 4.17: Mensaje informativo recibido tras la activación del riego.

Se han contemplado algunos errores potenciales, por ejemplo, que la diferencia entre dos sensores de humedad que monitorizan la misma zona de riego sea demasiado grande. Si dos sensores cercanos ofrecen valores razonablemente distintos, se considera que ha habido un error en la medición y se envía una notificación al usuario a través del bot de Telegram. No se contempla como un error ya que podría darse el caso que, por ejemplo, un cultivo necesite más agua que otro y que ambos estén en la misma zona. Por lo que las medidas de los sensores podrían verse afectadas.

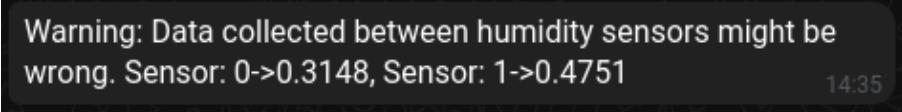


Figura 4.18: Mensaje de posible error enviado cuando la diferencia entre dos sensores de humedad de la tierra es significativa.

En el directorio `code/apps/actions/controllers` se encuentran el fichero de configuración de actuadores, `config_controllers.py`. Este fichero es análogo al de los sensores, en el se define la clase `Controller`, como se muestra a continuación:

---

```
# Config controllers file
class Controller:
    def __init__(self, pin, data):
        self.pin = pin
        self.data = data # In the bomb case, liters per minute

bomb = Controller(24, 4)
```

---

El propósito de este tipo de ficheros de configuración es el de poder añadir nuevos dispositivos de manera sencilla.

En este mismo directorio también se encuentra el fichero `controllers.py`, donde se definen las funciones para el apagado y encendido del actuador deseado. Estas funciones reciben el pin de los actuadores. En el caso de la bomba de riego se está utilizando el GPIO 24.

Estas funciones dan de alta el pin de salida deseado haciendo uso de la librería GPIO, con `GPIO.OUT` y luego encienden o apagan el dispositivo que está utilizando pasádole un valor

#### ***4.3. ARQUITECTURA E IMPLEMENTACIÓN SOFTWARE***

---

binario.

Al igual que con las funcionalidades descritas anteriormente, la aplicación de riego tiene sus propios logs. Estos logs se encuentran en *code/apps/actions/logs*. En este directorio se encuentran los ficheros *success.log*, *error.log* y si procede *cronerror.log*. La aplicación se ejecuta en Cron y se lanza una vez al día, a las 00:30.

# **Capítulo 5**

## **Casos de uso**

En este capítulo se incluyen ejemplos y casos de uso de las funcionalidades introducidas en el capítulo 4. Se pretende mostrar ejemplos del flujo de la información, desde que es recogida por los sensores y almacenada en la base de datos, hasta que es visualizada en la aplicación Dash o a través del bot de Telegram.

### **5.1. caso dedato bueno cambiar nombre**

En esta sección se muestran casos de uso y el flujo de funcionamiento de la plataforma completa. Como norma general que se repite para cada uno de los sensores disponibles, en primer lugar el sensor recoge la información y es guardada en la base de datos. Después, desde la aplicación Dash y desde el Bot de Telegram se puede acceder a la base de datos para visualizar la información. En las siguientes secciones se presentan ejemplos con valores para explicar de mejor manera el proceso.

#### **5.1.1. Temperatura y humedad ambiente**

El caso de la temperatura y la humedad ambiente es análogo. En primer lugar se realiza la recogida de datos usando los sensores *DHT22*. Los datos recogidos se suben a la colección *sensors\_data* de la base de datos *greenhouseDB*. En las figuras 5.1y 5.2 se muestran los datos almacenados de humedad y temperatura ambiente correspondientes al 6 de Junio de 2022, para la lectura realizada a las 4 de la mañana. En estas figuras se puede observar la estructura de los

## 5.1. CASO DEDATO BUENO CAMBIAR NOMBRE

documentos que representan la información sobre humedad y temperatura ambiente, que son similares al resto de datos que se manejan en este TFG. La información almacenada en cada campo se puede encontrar en la sección 4.3.1.

▼	8 Objectid("62a002f6d9c35d3e7ba53116")	{ 6 fields }	Object
└	_id	Objectid("62a002f6d9c35d3e7ba53116")	Objectid
└	type	ambient temperature	String
└	@@ value	20.0	Double
└	sensorID	22	Int32
└	location	outdoor	String
└	date	2022-06-08 04:01:26.276Z	Date
▼	9 Objectid("62a002f8d9c35d3e7ba53117")	{ 6 fields }	Object
└	_id	Objectid("62a002f8d9c35d3e7ba53117")	Objectid
└	type	ambient temperature	String
└	@@ value	19.7999992370605	Double
└	sensorID	27	Int32
└	location	indoor	String
└	date	2022-06-08 04:01:28.024Z	Date

Figura 5.1: Datos almacenados en *sensors\_data* para la temperatura ambiente.

▼	6 Objectid("62a002ebd9c35d3e7ba53114")	{ 6 fields }	Object
└	_id	Objectid("62a002ebd9c35d3e7ba53114")	Objectid
└	type	ambient humidity	String
└	@@ value	34.7999992370605	Double
└	sensorID	22	Int32
└	location	outdoor	String
└	date	2022-06-08 04:01:15.195Z	Date
▼	7 Objectid("62a002f4d9c35d3e7ba53115")	{ 6 fields }	Object
└	_id	Objectid("62a002f4d9c35d3e7ba53115")	Objectid
└	type	ambient humidity	String
└	@@ value	43.7999992370605	Double
└	sensorID	27	Int32
└	location	indoor	String
└	date	2022-06-08 04:01:24.531Z	Date

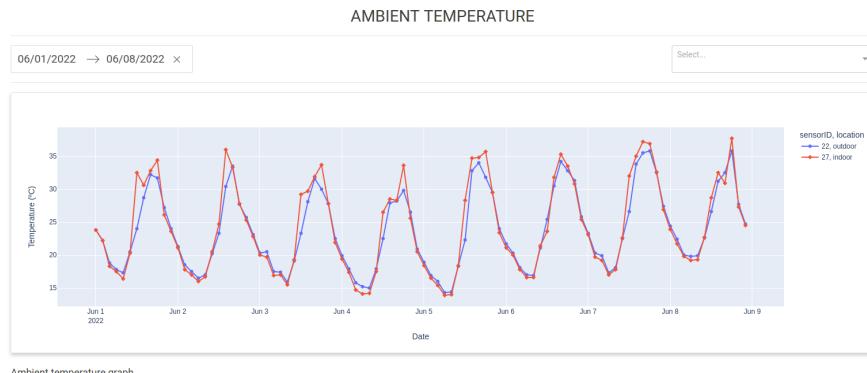
Figura 5.2: Datos almacenados en *sensors\_data* para la humedad ambiente.

Una vez los datos son almacenados en la base de datos pueden ser consultados desde las diferentes aplicaciones que se han desarrollado. En la figura 5.3 se observa la gráfica obtenida en la aplicación web de Dash que muestra los datos relativos a la temperatura ambiente recogidos desde el 1 hasta el 8 de Junio de 2022. Para obtener esta gráfica es necesario acceder al apartado *AMBIENT TEMPERATURE* dentro de *SENSORS DATA* del menú de la aplicación y seleccionar las fechas deseadas en el desplegable de la parte superior derecha diseñado para este uso (ver figura 5.4). También es posible obtener únicamente los datos de un sensor, para ello es necesario utilizar el desplegable de sensores, situado en la parte superior izquierda de la página (ver figura 5.5).

En las figuras se observa como las lecturas de ambos sensores miden aproximadamente lo mismo, esto confirma el correcto funcionamiento del sistema. En la época del año en la que se realizaron las lecturas el invernadero se encontraba abierto, de manera que la temperatura interior y exterior debe ser similar. Las fluctuaciones que se muestran en la figura se corresponden con las altas temperaturas durante el día y la bajada de estas durante la noche.

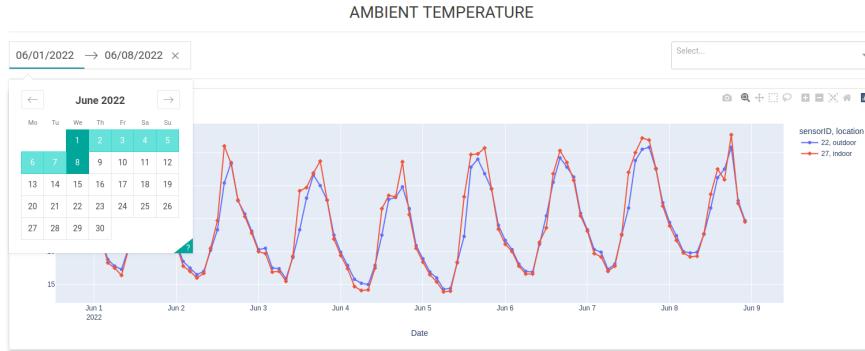
## 5.1. CASO DEDATO BUENO CAMBIAR NOMBRE

---



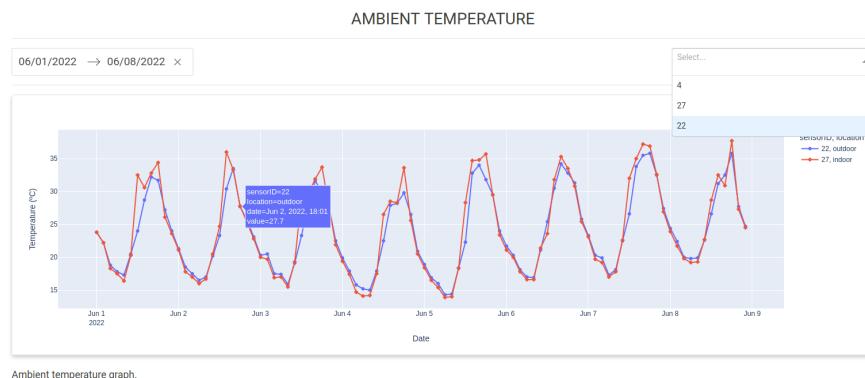
Ambient temperature graph.

Figura 5.3: Gráfica obtenida mediante la aplicación web con datos de temperatura ambiente desde 1 al 8 de Junio de 2022.



Ambient temperature graph.

Figura 5.4: Uso del selector de fechas en la aplicación web.



Ambient temperature graph.

Figura 5.5: Uso del selector de sensores en la aplicación web.

En la figura 5.5 se puede observar que se representan los sensores disponibles. En esta figura

## 5.1. CASO DEDATO BUENO CAMBIAR NOMBRE

se observa que hay tres sensores, esto se debe a que uno de los sensores fue cambiado de ID para comprobar el funcionamiento del sistema.

También es posible obtener esta gráfica haciendo uso del bot de Telegram mediante el comando */ambienttemperaturefig* con sus parámetros opcionales para indicar la fecha deseada *-startdate* y *-enddate*, como se muestra en la figura 5.6.

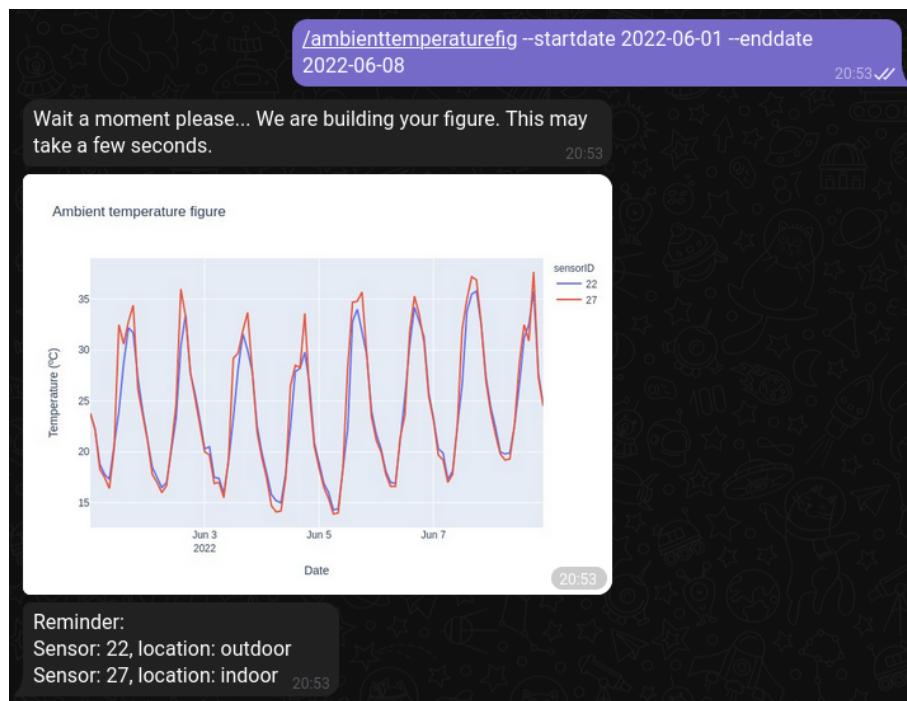


Figura 5.6: Gráfica obtenida mediante el bot de Telegram con datos de temperatura ambiente desde 1 al 8 de Junio de 2022.

Si se ejecuta el comando */ambienttemperaturefig* sin ningún argumento se obtiene la gráfica con los datos obtenidos por defecto como se muestra en la figura 5.7. Los datos por defecto son los datos de las últimas 24 horas. El comando permite también utilizar un único parámetro opcional, *-startdate*(ver figura 5.8). En el caso de usar únicamente el parámetro *-enddate* se envían los datos desde el primer valor almacenado hasta la fecha de fin indicada, como se muestra en la figura 5.9.

## 5.1. CASO DEDATO BUENO CAMBIAR NOMBRE

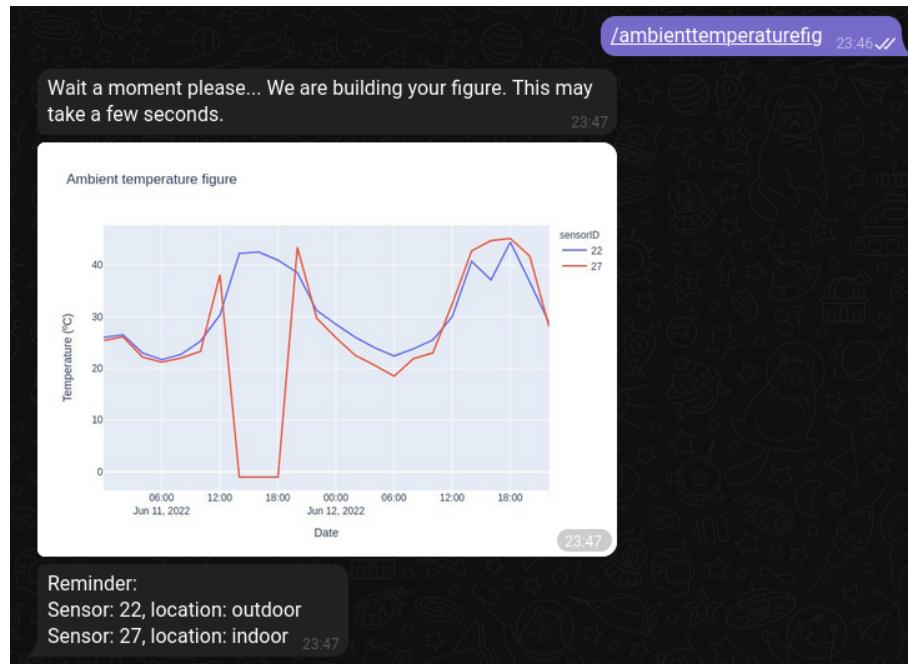


Figura 5.7: Gráfica obtenida mediante el comando */ambienttemperaturefig* realizado el día 12 de Junio de 2022.

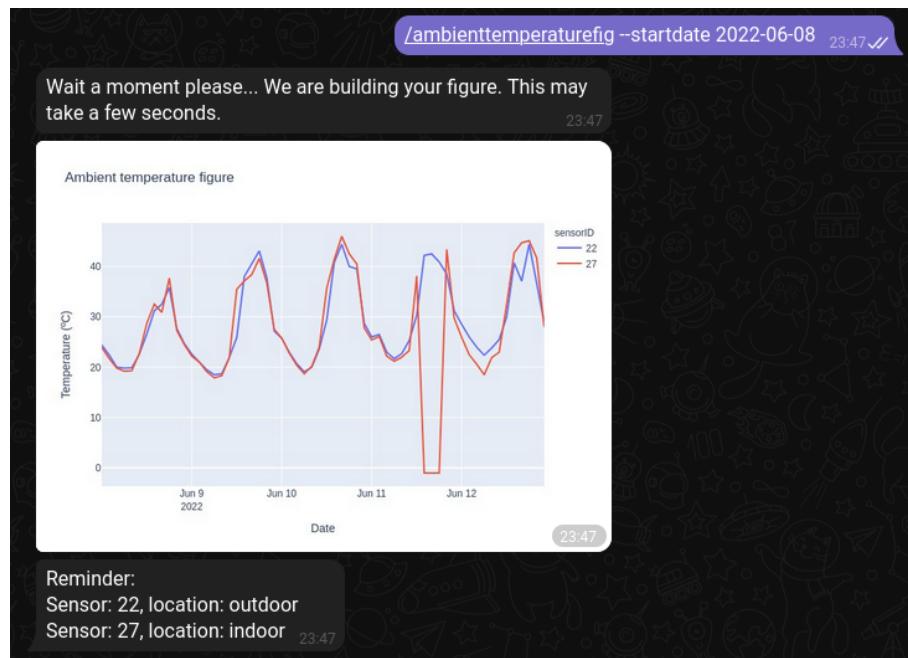


Figura 5.8: Gráfica obtenida mediante el comando */ambienttemperaturefig* realizado el día 12 de Junio de 2022.

## 5.1. CASO DEDATO BUENO CAMBIAR NOMBRE

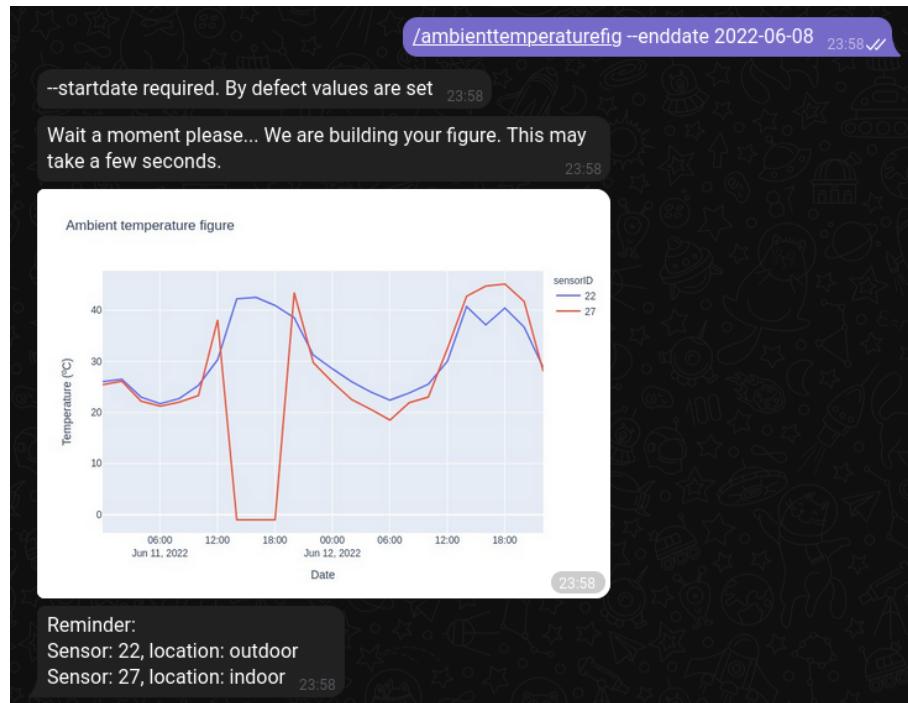


Figura 5.9: Gráfica obtenida mediante el comando `/ambienttemperaturefig` realizado el día 12 de Junio de 2022.

Como se observa en las figuras anteriores se puede observar de manera rápida y sencilla un error en la lectura de un sensor, el día 11 de Junio de 2022 el sensor 27 de temperatura falló en su lectura. Este tipo de errores como se indica en la sección 4.3.4 también se notifican al usuario de manera que no sea necesario estar consultando el bot para confirmar su correcto funcionamiento. En la siguiente figura se muestra el mensaje recibido tras el error de lectura.

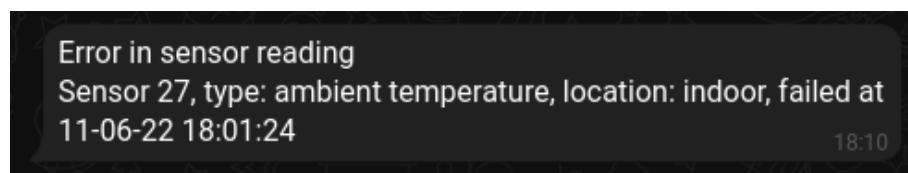


Figura 5.10: Mensaje de error obtenido el 11 de Junio de 2022 tras error de lectura de temperatura ambiente del sensor 27.

Otros de los comandos disponibles en el bot de Telegram relativos a la temperatura ambiente son `/ambienttemperature`, con el que se obtiene la temperatura ambiente en tiempo real (ver figura 5.11) y `/ambienttemperatedb` (ver figura 5.12) que devuelve la última lectura almacenada en la base de datos.

## 5.1. CASO DEDATO BUENO CAMBIAR NOMBRE

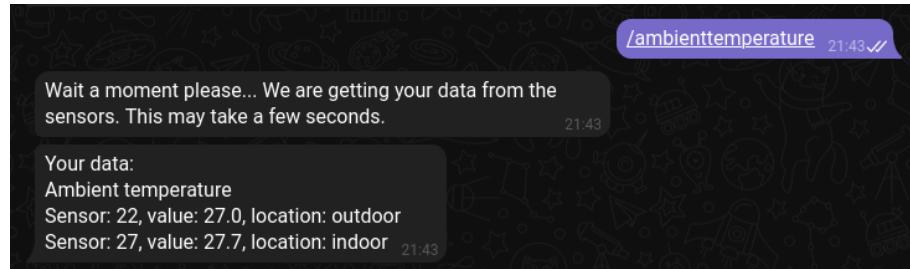


Figura 5.11: Mensaje recibido haciendo uso del comando */ambienttemperature*.

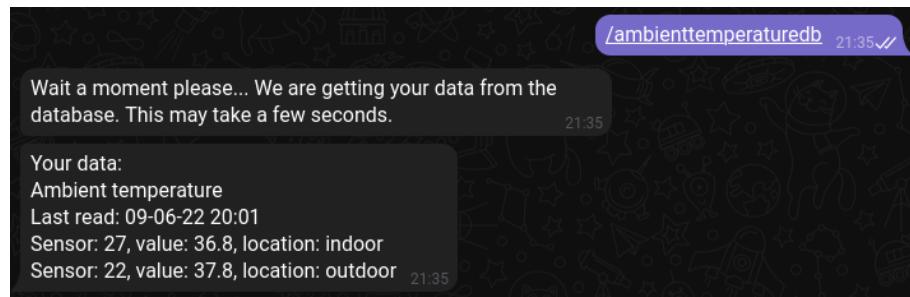


Figura 5.12: Mensaje recibido haciendo uso del comando */ambienttemperaturedb*.

A pesar de que en los mensajes anteriores se da información sobre los sensores que refieren-  
cian las figuras y los mensajes, el usuario puede utilizar el comando */sensorsinfo* para obtener  
información detallada sobre todos los sensores del sistema como se muestra en la figura 5.13.

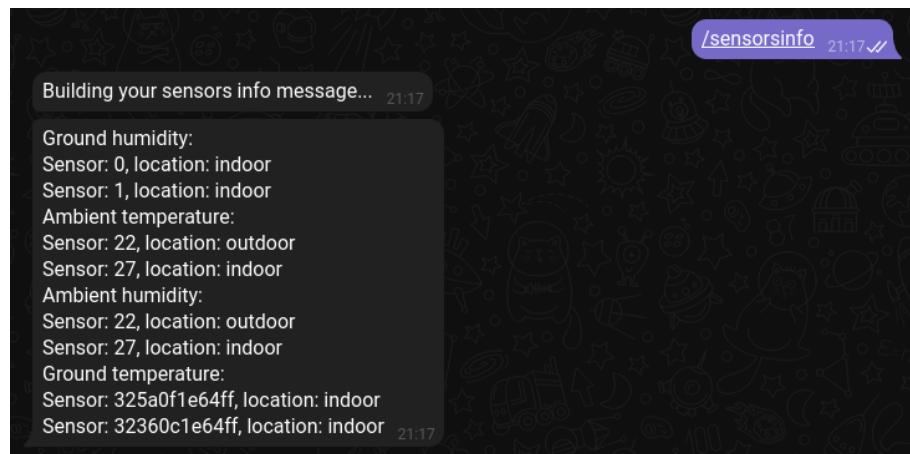


Figura 5.13: Mensaje recibido haciendo uso del comando */sensorsinfo*.

Como se ha mencionado anteriormente el caso de la humedad ambiente es análogo al de la temperatura. A continuación se muestran las diferentes figuras relativas a la humedad ambiente.

## 5.1. CASO DEDATO BUENO CAMBIAR NOMBRE

De igual manera en las figuras se pueden apreciar las fluctuaciones correspondientes al cambio de humedad en la noche respecto al cambio durante el día. Los valores obtenidos por ambos sensores también son similares en sus medidas en cuanto a la humedad.

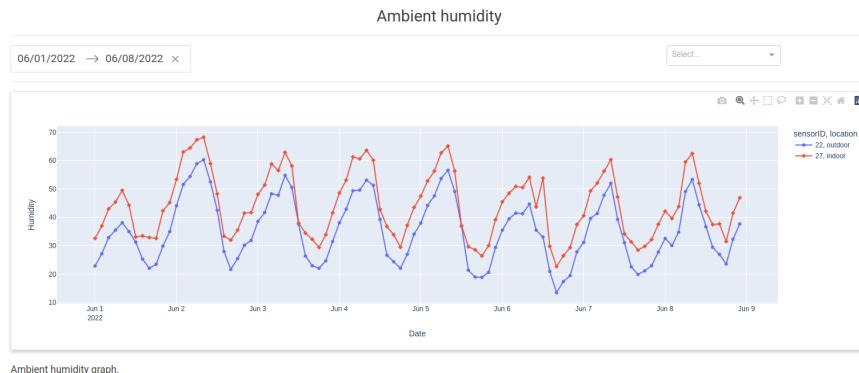


Figura 5.14: Gráfica obtenida mediante la aplicación web con datos de temperatura ambiente desde 1 al 8 de Junio de 2022.



Figura 5.15: Gráfica obtenida mediante el bot de Telegram con datos de temperatura ambiente desde 1 al 8 de Junio de 2022.

## 5.1. CASO DEDATO BUENO CAMBIAR NOMBRE

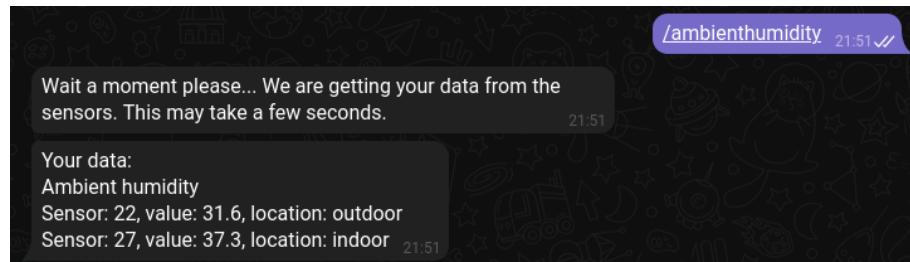


Figura 5.16: Mensaje recibido haciendo uso del comando */ambienthumidity*.

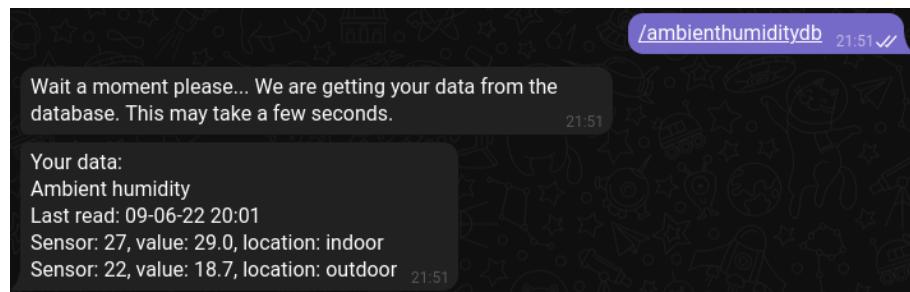


Figura 5.17: Mensaje recibido haciendo uso del comando */ambienthumiditydb*.

### 5.1.2. Temperatura de la tierra

En esta sección se muestra el caso de uso de la aplicación para la recogida y consulta de datos de temperatura de la tierra realizada mediante los sensores *DS18B20* que son subidos a la colección *sensors\_data* de la base de datos *greenhouseDB*. Es muy similar al descrito en la sección 5.1.1. La estructura seguida en la recogida de datos es la que se muestra en la figura 5.18



Figura 5.18: Arquitectura de los datos almacenados en *sensors\_data* para la temperatura de la tierra.

La gráfica en la aplicación web se puede obtener en *GROUND TEMPERATURE* dentro del desplegable *SENSORS DATA* del menú principal como se muestra en la figura 5.19. Por defecto, los datos mostrados son los correspondientes a la última semana.

## 5.1. CASO DEDATO BUENO CAMBIAR NOMBRE



Figura 5.19: Sección *GROUND TEMPERATURE* en la aplicación web de Dash

En la figura 5.20 se puede observar una diferencia clara en las medidas de las horas centrales del día. Esto se debe a que el sensor representado en azul esta colocado más cercano a la superficie de la tierra, por lo que recibe mucho más calor, sobre todo en días con altas temperaturas como es el caso de la figura. De igual manera se pueden apreciar las fluctuaciones de temperatura correspondientes al día y a la noche.

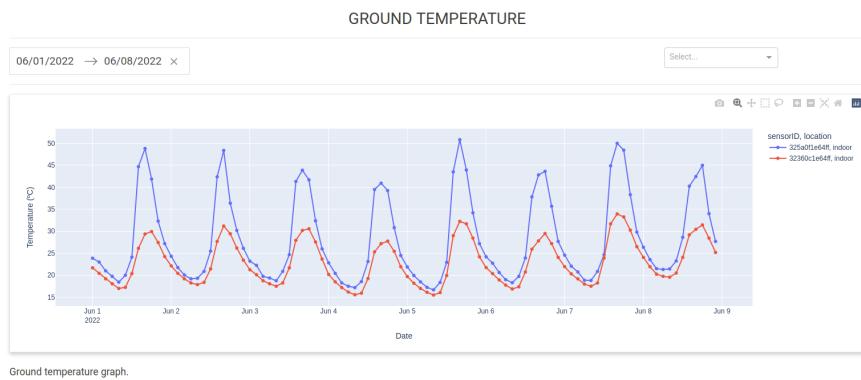


Figura 5.20: Gráfica obtenida mediante la aplicación web de Dash con datos de temperatura de la tierra desde 1 al 8 de Junio de 2022

En las sucesivas figuras se muestra el uso del bot de Telegram para obtener los datos relativos a la temperatura de la tierra.

Como se muestra en las figuras 5.21 y 5.22, la temperatura de la tierra de ambos sensores es bastante pareja en horas que no reciben la luz del sol. Por ejemplo, el comando de la figura 5.21 fue ejecutado a las 23:09.

Tal y como se puede observar en la figura 5.23 a medida que avanza el día la diferencia entre las lecturas de ambos sensores es mayor, en cambio, por la noche las mediciones son muy

## 5.1. CASO DEDATO BUENO CAMBIAR NOMBRE

similares.

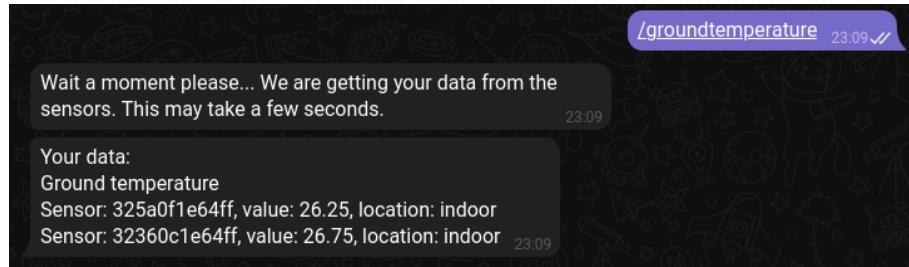


Figura 5.21: Mensaje recibido haciendo uso del comando */groundtemperature*

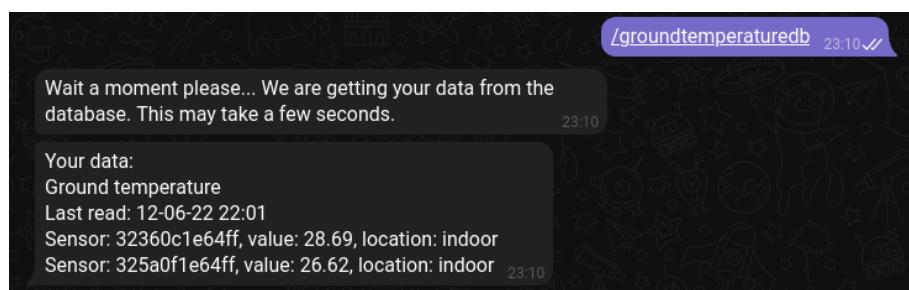


Figura 5.22: Mensaje recibido haciendo uso del comando */groundtemperatedb*

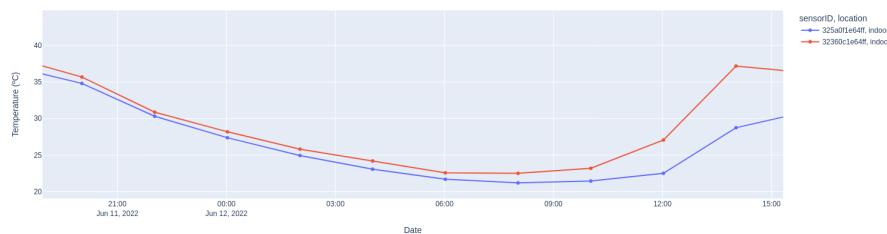


Figura 5.23: Diferencia entre lecturas de ambos sensores. Medidas tomadas entre las 20 del dia 11 de Junio de 2022 y las 14 del dia 12 de Junio de 2022

Igual que para el resto de magnitudes, es posible utilizar el bot de Telegram para obtener gráficas como se muestra en las figuras 5.24 y 5.25.

## 5.1. CASO DEDATO BUENO CAMBIAR NOMBRE

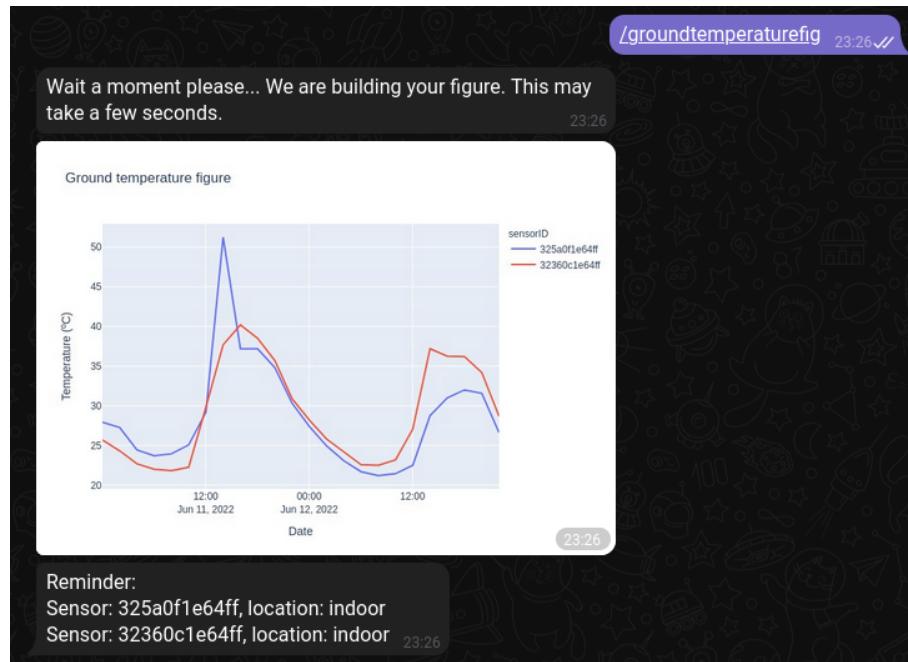


Figura 5.24: Gráfica obtenida mediante el comando `/groundtemperaturefig` realizado el día 12 de Junio de 2022.



Figura 5.25: Gráfica obtenida mediante el bot de Telegram con datos de temperatura ambiente desde 1 al 8 de Junio de 2022.

### 5.1.3. Humedad de la tierra

En esta sección se presenta el tratamiento de la información de la humedad de la tierra. En primer lugar, se realiza la lectura de datos utilizando sensores YL-69. La lectura de estos higrómetros es almacenada en la base de datos *greenhouseDB*, en concreto en la colección *sensors\_data*. La estructura seguida en la recogida de datos de este tipo es la que se muestra en la figura 5.26



Figura 5.26: Arquitectura de los datos almacenados en *sensors\_data* para la humedad de la tierra.

Estos datos al igual que en los casos anteriores pueden ser consultados o bien desde la aplicación web, obteniéndola en *GROUND HUMIDITY* dentro del desplegable *SENSORS DATA* del menú principal, o bien desde el bot de Telegram. Todas las figuras, comandos y datos disponibles para la humedad de la tierra son análogos a los casos descritos en las secciones 5.1.1 y 5.1.2.

La humedad de la tierra está relacionada con la aplicación de riego que se ha desarrollado. Como se expone en la sección 4.3.5. En las figuras sucesivas se muestra como la relación entre los datos de humedad de la tierra y los datos almacenados en la colección *controllers\_data* conservan una estrecha relación. En las figuras 5.27 y 5.28 se muestra como a mayor sequedad de la tierra el sistema riega con una mayor cantidad de agua. El terreno está más seco cuando la lectura de los sensores es más cercana a uno, es por este motivo que el día 4 de Junio se regó con una mayor cantidad de agua que el día 6.

## 5.1. CASO DEDATO BUENO CAMBIAR NOMBRE



Irrigation data graph.

Figura 5.27: Gráfica con datos de riego del 4 al 6 de junio de 2022.

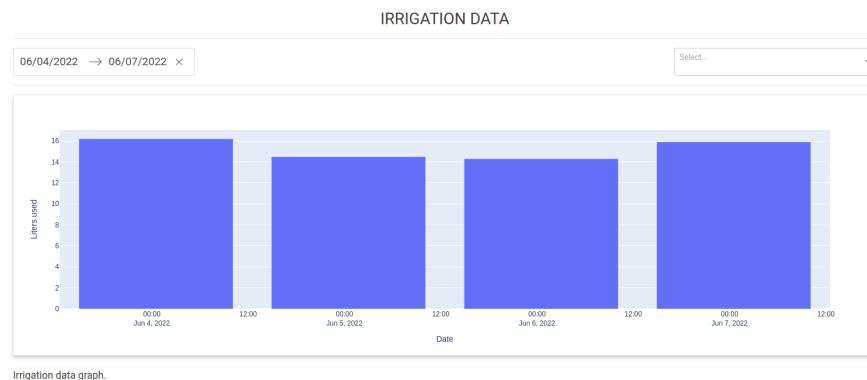


Ground humidity graph.

Figura 5.28: Gráfica con datos de humedad de la tierra del 4 al 6 de junio de 2022.

Las gráfica para los actuadores, en este caso las gráfica de riego, se generan siguiendo el mismo proceso que para los sensores. Se puede utilizar la aplicación web, estas figuras se encuentran en el desplegable *IRRIGATION* de la sección *CONTROLLERS DATA* del menú principal (ver figura 5.29). Haciendo uso del comando */irrigationfig*, con sus parámetros opcionales *-startdate* y *-enddate*, se obtiene la figura de riego y con el comando */irrigationdb* se obtiene el último dato almacenado en la base de datos sobre el riego. El comando */controllersinfo* devuelve un mensaje con información relevante sobre los diferentes actuadores del sistema, en este caso, sobre la bomba de riego (ver figura 5.31).

## 5.1. CASO DEDATO BUENO CAMBIAR NOMBRE



Irrigation data graph.

Figura 5.29: Datos de riego obtenidos mediante la aplicación web del 04 al 07 de junio de 2022.

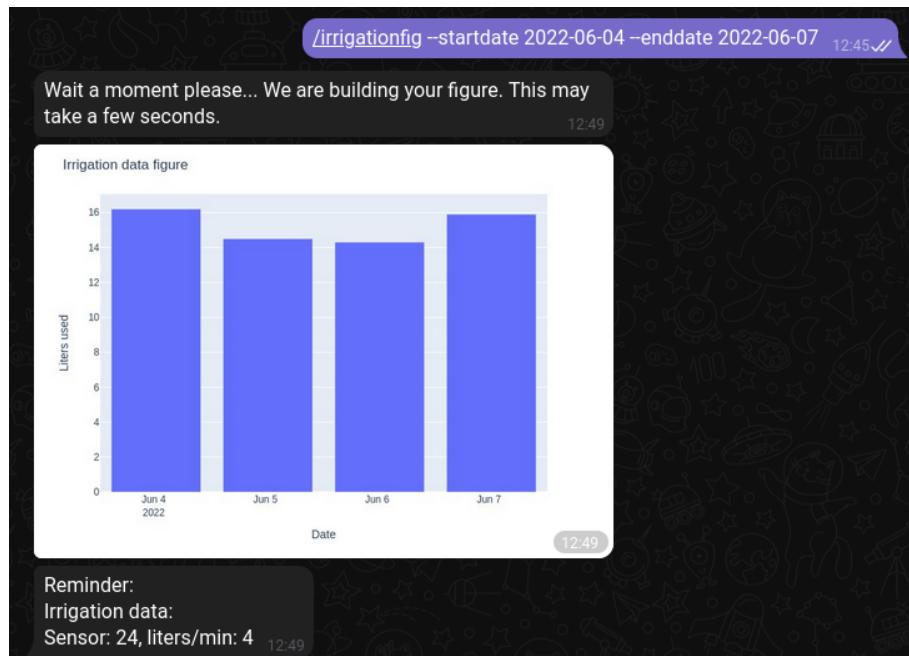


Figura 5.30: Datos de riego obtenidos mediante el bot de Telegram del 04 al 07 de junio de 2022.

Como se muestra en las figuras 5.29 y 5.30 las gráficas elegidas para mostrar los datos relativos al riego son barras que representan la cantidad de litros usados para el riego en un día. En la figura 5.31 se observa la salida del comando */controllersinfo*. Este comando proporciona información sobre los actuadores del sistema, en el caso de la bomba se indica su ID (el pin al que está conectado) y su caudal.

## 5.2. COSAS PARTICULARES CAMBIAR NOMBRE

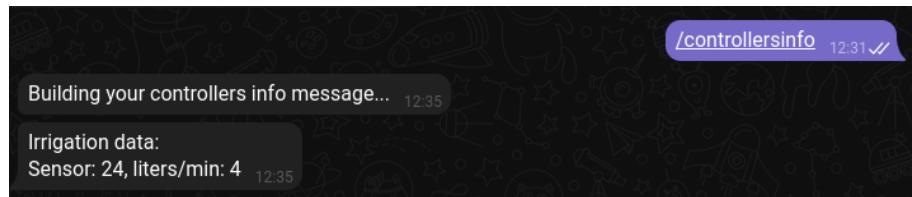


Figura 5.31: Salida del comando `/controllersinfo`.

## 5.2. cosas particulares cambiar nombre

Como se menciona en el apartado 4.3.3, uno de los objetivos de la aplicación web es el de poder realizar un análisis concreto del estado del sistema y encontrar así errores. La aplicación es útil para detectar comportamientos anormales en los sensores o actuadores. Por ejemplo, en la figura 5.32 se observa que durante varios días el sensor tomaba medidas muy cercanas a 1, esto permitió comprobar que uno de los sensores de humedad del suelo estaba fallando.

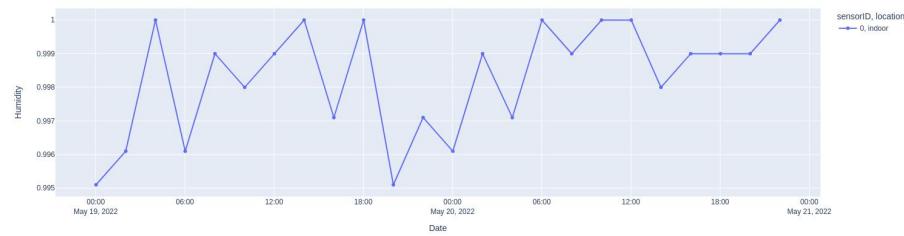


Figura 5.32: Gráfica que muestra un comportamiento anormal de uno de los higrómetros.

Otro caso de error se muestra en la figura 5.33, donde se muestra que el sensor con ID 22 no estaba funcionando correctamente.

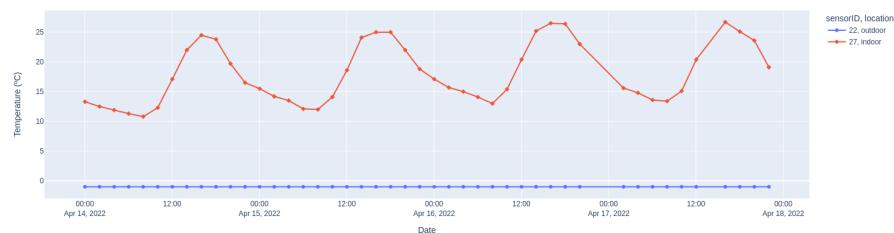


Figura 5.33: Gráfica que muestra un comportamiento anormal de uno de los sensores de temperatura ambiente.

## 5.2. COSAS PARTICULARES CAMBIAR NOMBRE

---

Otros de los objetivos del control de errores es el de notificar al usuario al momento de su detección. Para ello se envían mensajes cuando esta clase de errores es detectada, como se muestra en los apartados anteriores.

En las secciones 5.1.1, 5.1.2 y 5.1.3 se muestran las distintas posibilidades que existen para la obtención de información para ese tipo de datos en concreto, pero también es posible realizar este análisis usando los comandos `/sensorsdata`, `/sensorsdata` y `/getallfigures`. Con estos comandos se puede obtener datos de todos los tipos de sensores a la vez, en lugar de obtenerlos para cada tipo de sensor individualmente. Sirven para realizar un análisis del estado del sistema más general.

Cuando el bot recibe el comando `/getallfigures` se envían todas las figuras disponibles y un mensaje adicional indicando que ya se ha terminado el proceso (ver figuras 5.36, 5.37 y 5.38). Se observa que todas las figuras tienen lógica con lo comentado en los apartados anteriores y que el sistema se encuentra funcionando correctamente.

Las gráficas se generan siguiendo el mismo proceso que el explicado en la sección 5.1, sólo que en lugar de ser sobre un grupo de sensores, se generan las gráficas de todos los sensores disponibles.

La salida de los comandos `/sensorsdata` y `/sensorsdata` que se muestran en las figuras 5.34 y 5.35 también es coherente.

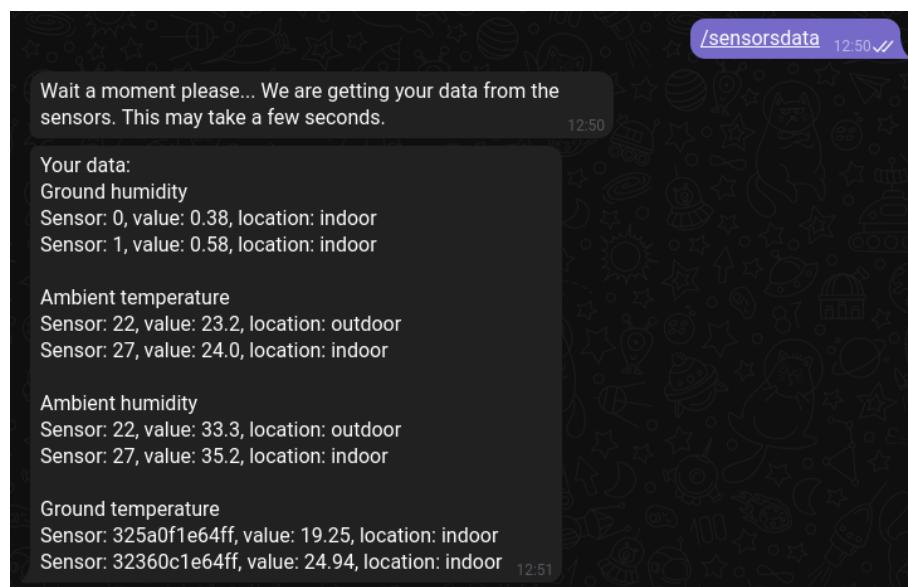
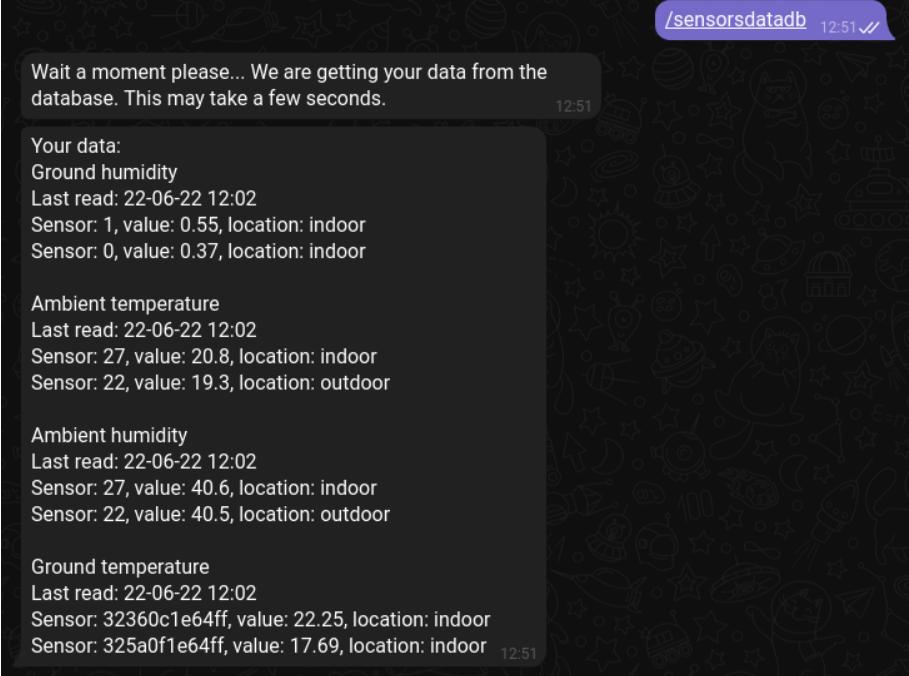


Figura 5.34: Salida del comando `/sensorsdata`.

## 5.2. COSAS PARTICULARES CAMBIAR NOMBRE

---



The screenshot shows a mobile application interface with a dark background featuring a pattern of various icons related to space and technology. At the top right, there is a blue rounded rectangle containing the text '/sensorsdataadb' and the time '12:51' followed by a checkmark icon.

Below this, a message bubble displays the text: 'Wait a moment please... We are getting your data from the database. This may take a few seconds.' with the timestamp '12:51' at the bottom right of the message area.

The main content area contains several sections of sensor data:

- Your data:**
  - Ground humidity
  - Last read: 22-06-22 12:02
  - Sensor: 1, value: 0.55, location: indoor
  - Sensor: 0, value: 0.37, location: indoor
- Ambient temperature**
  - Last read: 22-06-22 12:02
  - Sensor: 27, value: 20.8, location: indoor
  - Sensor: 22, value: 19.3, location: outdoor
- Ambient humidity**
  - Last read: 22-06-22 12:02
  - Sensor: 27, value: 40.6, location: indoor
  - Sensor: 22, value: 40.5, location: outdoor
- Ground temperature**
  - Last read: 22-06-22 12:02
  - Sensor: 32360c1e64ff, value: 22.25, location: indoor
  - Sensor: 325a0f1e64ff, value: 17.69, location: indoor

Figura 5.35: Salida del comando /sensorsdataadb.

## 5.2. COSAS PARTICULARES CAMBIAR NOMBRE

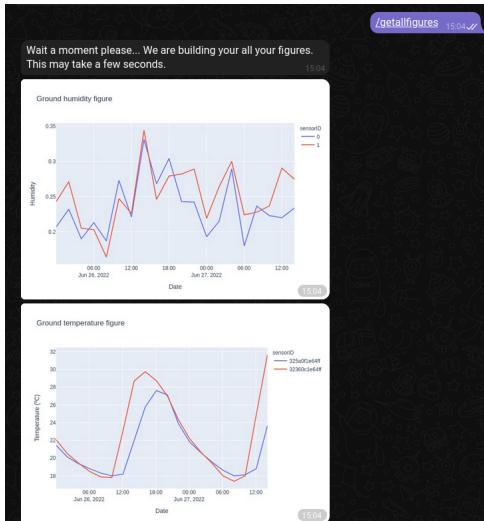


Figura 5.36: Primera parte salida del comando `/getallfigures`.

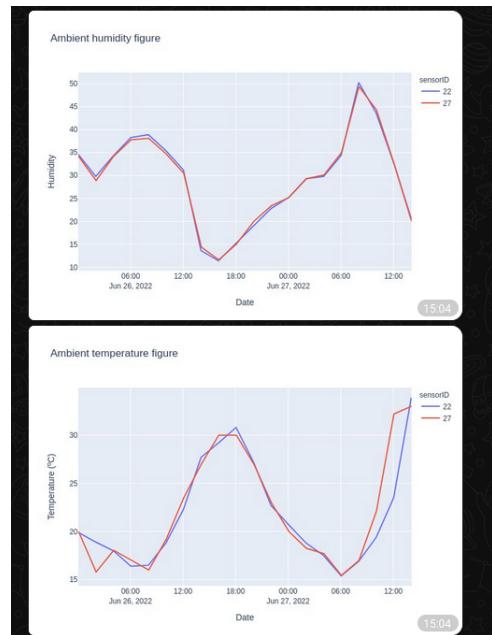


Figura 5.37: Segunda parte salida del comando `/getallfigures`.

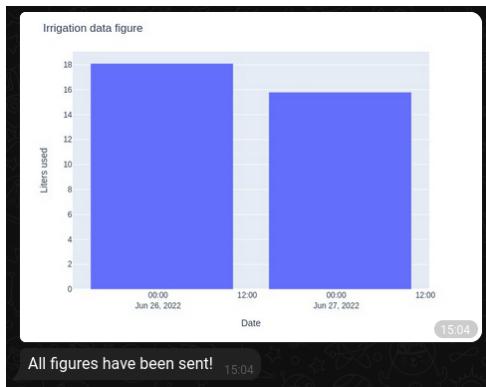


Figura 5.38: Tercera parte salida del comando `/getallfigures`.

# **Capítulo 6**

## **Conclusiones**

### **6.1. Conclusiones finales**

una conclusión

### **6.2. Trabajos futuros**

trabajo futuros

# **Apéndice A**

## **Manual de usuario**

Si has creado una aplicación, siempre viene bien tener un manual de usuario. Pues ponlo aquí.

# Bibliografía

- [1] HISTORIA DE LA ROBÓTICA. <https://scielo.isciii.es/pdf/aue/v31n3/v31n3a02.pdf>
- [2] DOCUMENTACIÓN OFICIAL DE JAVASCRIPT <https://developer.mozilla.org/es/docs/Web/JavaScript>
- [3] DOCUMENTACION OFICIAL DE DJANGO <https://docs.djangoproject.com>
- [4] DOCUMENTACION OFICIAL DE DJANGO-CHANNELS <https://channels.readthedocs.io/en/stable/>
- [5] DOCUMENTACION OFICIAL DE DOCKER <https://docs.docker.com/>
- [6] DOCUMENTACIÓN OFICIAL DE WEBRTC. <https://webrtc.org/>
- [7] PABLO MORENO. TORNEOS DE PROGRAMACIÓN DE ROBOTS EN UNA PLATAFORMA ONLINE. (2020) [https://gsyc.urjc.es/jmplaza/students/tfm-kibotics-torneos-pablo\\_moreno-2020.pdf](https://gsyc.urjc.es/jmplaza/students/tfm-kibotics-torneos-pablo_moreno-2020.pdf)
- [8] ÁLVARO PANIAGUA. SIMULADOR DE ROBOTS CON TECNOLOGÍAS WEB VR. (2018) [https://github.com/RoboticsLabURJC/2018-tfg-alvaro\\_paniagua](https://github.com/RoboticsLabURJC/2018-tfg-alvaro_paniagua)
- [9] DAVID ROLDÁN, SAKSHAY MAHNA JOSÉ M. CAÑAS INTERNATIONAL CONFERENCE ON ROBOTICS IN EDUCATION (RIE-2021), PP 243-255, ADVANCES IN INTELLIGENT SYSTEMS AND COMPUTING, VOL 1359. SPRINGER, 2022. <https://gsyc.urjc.es/jmplaza/papers/rie2021-unibotics-draft.pdf>