

UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS

Decana de América, Universidad del Perú

FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

Escuela Académica Profesional de Ingeniería de Sistemas



Sistema de Riego Inteligente

Curso

Internet de las Cosas

Estudiantes

Curay Chacón Piero Yahir – 100%

Mejía Benito Elphy Xavier – 100%

Pérez Pijo Kevin Gersy – 100%

Pimentel Yanac Manuel Fernando – 100%

Quispe Ñaupa Wiliam Luis – 100%

Tello Palacios Franco Khaled – 100%

Docente

Yéssica Rosas Cuevas

Agosto, 2022

ÍNDICE

1. INTRODUCCIÓN	1
1.1. Revisión del estado del arte	2
1.1.1. Sistema de riego autónomo de bajo costo para expansión de área agrícola en laderas de los valles del sur del Perú basado en IOT	2
1.1.2. Diseño de una solución basada en el internet de las cosas (IoT) empleando Lorawan para el monitoreo de cultivos agrícolas en Perú	3
1.1.3. Automatización y telecontrol del sistema de riego para las áreas verdes del templo Santos de los Últimos Días - Arequipa	4
1.2. Formulación del problema de investigación.....	5
1.2.1 Problema General.....	5
1.2.2 Problemas Específicos.....	5
1.3. Objetivos de la investigación.....	6
1.3.1 Objetivo General.....	6
1.3.2 Objetivos Específicos	6
2. MARCO TEÓRICO.....	7
2.1. Dracaena	7
2.2. Sistema de riego inteligente	7
2.3. Sistema de riego manual	8
3. COMPONENTES DEL SISTEMA DE RIEGO	9
3.1. Microcontrolador ESP-8266	9
3.2. Broker MQTT	9
3.3. Eclipse Mosquitto.....	10
3.4. Node-Red.....	11
3.5. XAMPP - Apache	12
3.6. PhpMyAdmin	13
3.7. Componentes del sistema de riego	13

4. IMPLEMENTACIÓN DEL SISTEMA	15
4.1. Requisitos del sistema.....	15
4.1.1. Requisitos funcionales.....	15
4.1.2. Requisitos no funcionales.....	16
4.2. Arquitectura	16
4.3. Diagrama Esquemático del Circuito de Riego en Wokwi	17
4.4. Diagrama de clase.....	17
4.5. Metodología del sistema	17
5. Despliegue de APP dashboard con python-dash	30
5.1. Herramientas utilizadas.....	30
5.2. Preparación de un entorno virtual.....	31
5.3 Generación de API.....	32
5.4. Desarrollo del APP-Dashboard.....	33
5.5 Despliegue en Heroku	34
5.6. Resultado del despliegue.....	35
6. RESULTADOS	37
CONCLUSIONES	42
BIBLIOGRAFÍA	43

ÍNDICE DE ILUSTRACIONES

<i>Ilustración 1</i>	<i>Grupo de Investigación de Internet de las Cosas, UNMSM</i>	<i>1</i>
<i>Ilustración 2</i>	<i>Estructura del MQTT</i>	<i>10</i>
<i>Ilustración 3</i>	<i>Logo del Mosquito</i>	<i>11</i>
<i>Ilustración 4</i>	<i>Logo del Xampp</i>	<i>13</i>
<i>Ilustración 5</i>	<i>Logo del phpMyAdmin</i>	<i>13</i>
<i>Ilustración 6</i>	<i>Arquitectura del sistema de riego de la dracanea</i>	<i>16</i>
<i>Ilustración 7</i>	<i>Arquitectura del sistema de riego de la dracanea</i>	<i>17</i>
<i>Ilustración 8</i>	<i>Arquitectura del sistema de riego de la dracanea</i>	<i>17</i>
<i>Ilustración 8</i>	<i>Funcionamiento de la placa</i>	<i>18</i>
<i>Ilustración 9</i>	<i>Funcionamiento del sensor de la humedad</i>	<i>18</i>
<i>Ilustración 10</i>	<i>Funcionamiento del sensor DHT22</i>	<i>19</i>
<i>Ilustración 11</i>	<i>Implementación y funcionamiento del Broker MQTT en Windows</i>	<i>19</i>
<i>Ilustración 12</i>	<i>Funcionamiento del Broker MQTT por consola</i>	<i>20</i>
<i>Ilustración 13</i>	<i>Estructura del node mqtt</i>	<i>21</i>
<i>Ilustración 14</i>	<i>Estructura del node chart</i>	<i>22</i>
<i>Ilustración 15</i>	<i>Estructura del node join</i>	<i>22</i>
<i>Ilustración 16</i>	<i>Estructura del node debug</i>	<i>23</i>
<i>Ilustración 17</i>	<i>Estructura del node function</i>	<i>23</i>
<i>Ilustración 18</i>	<i>Estructura del node mysql</i>	<i>24</i>
<i>Ilustración 19</i>	<i>Estructura de los nodos de dashboard en el Node-Red</i>	<i>25</i>
<i>Ilustración 20</i>	<i>Estructura del sistema de riego en el Node-Red</i>	<i>26</i>
<i>Ilustración 21</i>	<i>Dashboard de los sensores en tiempo real</i>	<i>27</i>
<i>Ilustración 22</i>	<i>Estructura del dashboard en el Node-Red</i>	<i>28</i>
<i>Ilustración 23</i>	<i>Captura de las columnas en la base de datos en el PhpMyAdmin</i>	<i>28</i>
<i>Ilustración 24</i>	<i>Captura de los registros en la base de datos</i>	<i>29</i>
<i>Ilustración 25</i>	<i>proyecto en Visual Studio Code</i>	<i>31</i>
<i>Ilustración 26</i>	<i>Extracto del dataframe de los resultados revisados</i>	<i>32</i>

<i>Ilustración 27 Conexión del repositorio del api en Heroku</i>	34
<i>Ilustración 28 Conexión del repositorio de la app en Heroku</i>	35
<i>Ilustración 29 Resultados de api desplegado</i>	35
<i>Ilustración 30 Valores de temperatura del 26/08/2022</i>	36
<i>Ilustración 31 Series de tiempo del 26/08/2022</i>	36
<i>Ilustración 32 Portada general</i>	38
<i>Ilustración 33 Serie de temperatura</i>	38
<i>Ilustración 34 Serie de temperatura ambiental</i>	38
<i>Ilustración 35 Acercamiento gráfico</i>	39
<i>Ilustración 36 Organización de los componentes y sensores del sistema de riego</i>	39

ÍNDICE DE TABLA

<i>Tabla 1 Listado de nodos del Node-RED</i>	11
<i>Tabla 2 Listado de componentes utilizados</i>	13

1. INTRODUCCIÓN

En la actualidad, el Ministerio de Desarrollo Agrario y Riego del Perú destacó algunos aspectos relevantes para la agricultura relacionados con bajas eficiencias de riego que producen problemas de drenaje y salinidad, cultivos de la costa que exigen alta demanda de agua, erosión en los suelos de la sierra por prácticas agronómicas realizadas y tarifas de agua muy elevadas, que no cubren los costos de operación y mantenimiento de los sistemas de riego.

En 2018, un grupo de investigación de Internet de las Cosas de la Universidad Nacional Mayor de San Marcos presentó un sistema de riego inteligente de parques públicos de Lima basado en soluciones como un sistema de riego de huerta automatizado por Arduino en la Universidad Pública de Navarra, España; y un sistema inteligente usando redes de sensores inalámbricos en Nueva York. Actualmente, el sistema se gestiona a través de la tecnología LoRAWAN y SIGFOX, y un programa de riego que recoge información del clima de la zona para determinar cuándo y cuánto regar.

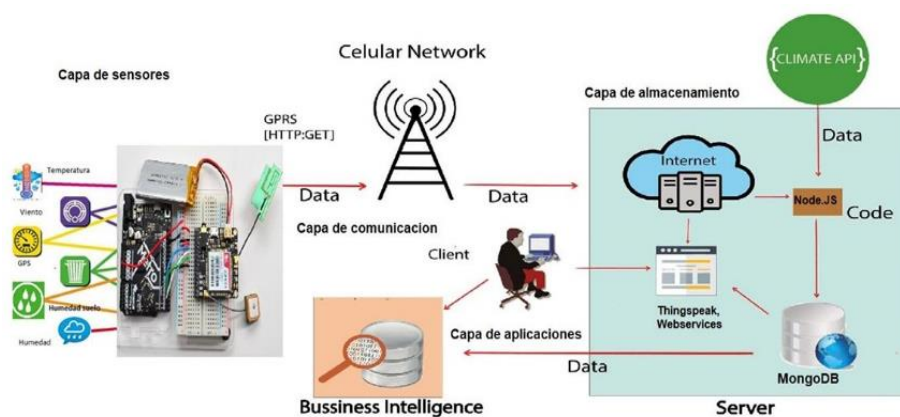


Ilustración 1 Grupo de Investigación de Internet de las Cosas, UNMSM

Según el informe Global Water Trends 2021 de Idrica, uno de los principales avances en el sector agrícola será la mejora de los recursos hídricos. El reporte indica que

las mejoras en el riego inteligente impulsarán el desarrollo del sector agrícola, el cual avanza hacia la gestión centralizada y automática.

1.1. Revisión del estado del arte

1.1.1. Sistema de riego autónomo de bajo costo para expansión de área agrícola en laderas de los valles del sur del Perú basado en IOT

- Autor y año: Juan Carlos Mercado García (2020)
- Problemática: La región sur del país cuenta con zonas agrícolas de producción, pero estas son pequeñas si hacemos una comparación con el norte del país. Las casusas que tenga poca área agrícola en la región sur se deben al poco aprovechamiento que se les da a las laderas y el poco uso de sistemas tecnológicos.
- Objetivos: Desarrollar un prototipo de sistema de riego por goteo autónomo de bajo costo para la expansión del área agrícola en el valle de Moquegua, en el fundo de Omo, basado en internet de las cosas (IoT).
- Población o muestra: El valle de Moquegua, en el fundo de Omo
- Resultados: Los resultados muestran que el área de cultivo que se logró generar fue de 330m² aproximadamente utilizando el prototipo de riego, con 3 líneas de riego y 45 árboles frutales plantados (manzanos, uvas, pepinos).
- Conclusiones: Se logró desarrollar un prototipo de riego de bajo costo que nos sirve para la expansión del área agrícola en laderas, probado en el valle de Moquegua, utilizando una plataforma de internet de las cosas.
- Contribución: Esta tesis da un ejemplar más en la aplicación de IOT al ámbito agrícola, específicamente a la expansión del área agrícola en los valles del sur del Perú.

1.1.2. Diseño de una solución basada en el internet de las cosas (IoT) empleando

Lorawan para el monitoreo de cultivos agrícolas en Perú

- Autor y año: Samuel Aguilar Zavaleta (2020)
- Problemática: Basándose en las características del estado de producción agrícola actual, del ambiente rural donde se desarrolla y haciendo uso de las técnicas de determinación de problemas correspondientes al marco lógico establecidos por la Comisión Económica para América Latina y el Caribe-CEPAL (Ortegón, Pacheco, & Prieto, 2015); se estableció el problema general de la investigación: “¿Cómo incrementar los niveles de monitoreo y control de los parámetros de producción de los cultivos agrícolas en el distrito de Pachacútec, provincia de Ica, departamento de Ica?”.
- Objetivos: Diseñar una solución basada en el Internet de las Cosas (IoT) empleando LoRaWAN para incrementar en 30% los niveles de monitoreo y control de los parámetros de producción de los cultivos agrícolas en el distrito de Pachacútec, provincia de Ica, departamento de Ica.
- Población o muestra: Distrito de Pachacútec, provincia de Ica, departamento de Ica.
- Resultados: Se logró desarrollar un prototipo de riego de bajo costo que nos sirve para la expansión del área agrícola en laderas, probado en el valle de Moquegua, utilizando una plataforma de internet de las cosas.
- Conclusiones: El sistema IoT planteado logra monitorear 5 parámetros de producción agrícola (temperatura del suelo, humedad del suelo, pH, dióxido de carbono y conductividad eléctrica), logra que el tiempo de entrega de la información se produzca en un máximo de una hora, la obtención de las mediciones se realiza de manera automatizada sin la intervención de personal,

la información se almacena en una nube computacional durante 5 años y el reporte de los cultivos se visualiza en un aplicativo móvil detallando la información numérica, gráficos, control de sistema de riego y bases de datos.

- Contribución: Esta tesis nos muestra otra forma de aplicar IoT empleando el protocolo de red Lorawan a la producción de los cultivos agrícolas en el distrito de Pachacútec del departamento de Ica mediante el monitoreo y evaluación de la condición del suelo.

1.1.3. Automatización y telecontrol del sistema de riego para las áreas verdes del templo Santos de los Últimos Días - Arequipa

- Autor y año: Ronald Diego Yalle Arce (2021)
- Problemática: El templo Santos de los Últimos Días presenta un sistema de riego presurizado operado de forma manual, con este nivel de implementación también se generan excesos en consumos de agua, en cantidad de personal, y en tiempo de operación reflejándose en altos costos en el tiempo.
- Objetivos: Automatizar y telecontrolar el sistema de riego para las áreas verdes en el Templo Santos de los Últimos Días con sede en la ciudad de Arequipa.
- Población o muestra: Templo Santos de los Últimos Días con sede en la ciudad de Arequipa
- Resultados: El diseño agronómico determinó los parámetros de operación para coberturas vegetales arbóreas, arbustivas, herbáceas, tapizantes y césped, garantizando la reposición de la lámina de agua para la especie más demandante (césped) y para los meses más críticos (abril - noviembre). Por su parte el diseño hidráulico garantizó presiones y caudales para los turnos de

riego permitiendo una adecuada uniformidad de riego (variaciones de presiones máximas del 20% en las subunidades de riego que significan variaciones de caudales menores del 10%).

- Conclusiones: La automatización mediante el controlador LXD (monocable) y decodificadores resultó ser el más adecuado teniendo en cuenta las 55 válvulas a controlar, ofrece mayor practicidad en la instalación y mantenimiento a través de los decodificadores, es un sistema flexible al crecimiento e incremento de válvulas sin tender nuevos cables al controlador y permite gestionar el riego de forma remota mediante la adición de cartuchos de comunicación de red.
- Contribución: Esta tesis nos muestra una forma de aplicar IoT en el ámbito agrícola, en este caso es la automatización de los sistemas de riego en el templo Santos de los Últimos Días en la ciudad de Arequipa

1.2. Formulación del problema de investigación

1.2.1 Problema General

¿Cómo influye el uso de los sistemas de riego inteligentes en las necesidades hídricas del cultivo dracaena?

1.2.2 Problemas Específicos

- ¿Qué conocimientos se requieren para crear un sistema de riego inteligente?
- ¿Qué tecnologías se pueden integrar en un sistema de riego inteligente?
- ¿A qué tipo de cultivos está orientado el sistema de riego inteligente?

1.3. Objetivos de la investigación

1.3.1 Objetivo General

- Realizar un prototipo de sistema de riego inteligente que satisfaga las necesidades hídricas del cultivo *Dracaena Deremensis*.

1.3.2 Objetivos Específicos

- Aplicar los conocimientos de internet de las cosas para el desarrollo del prototipo inteligente.
- Integrar tecnologías como Arduino, Node-red, phpMyAdmin y Python en el prototipo inteligente.
- Monitorear el estado del cultivo una vez se haya desplegado el prototipo inteligente.

2. MARCO TEÓRICO

2.1. Dracaena

Es un arbusto que tiene uno o varios troncos, usualmente de crecimiento lento. Es popular por la facilidad de cultivo y ornamentación. Tienen una gran capacidad para purificar el aire en espacios interiores y mejorar el ambiente. Su crecimiento y mantenimiento es a 15 grados aproximadamente y su riego puede ser irregular, lo que es cómodo su producción, pero si es importante que se mantenga en suelo húmedo. (Picand, 2015)

Tal como menciona Sun (2019), por el hecho de que suele necesitar pocos cuidados, es muy común su uso en oficinas o lugares donde con solo luz y calor puede tener un desarrollo adecuado. El suelo que requieren para crecer debe ser húmedo, con un PH entre 6 y 6.5. Si existe mucha acidez las hojas pueden sufrir de clorosis, y en caso contrario, si el suelo es demasiado alcalino pueden existir deficiencias de absorción de hierro. La humedad ambiente con la que la Dracaena crece de manera adecuada es entre el 60% y 100%.

2.2. Sistema de riego inteligente

Permiten la operación autónoma de la irrigación valiéndose de la tecnología, sin necesidad de contar con personal para llevarlo a cabo. Como menciona Gózales (2017) se trata de un sistema de riego que provee de agua a los cultivos de manera automatizada y que emplea normalmente la aspersión o el goteo. Existen sistemas de riego automático que combinan tanto la aspersión como el goteo y que permiten combinar las ventajas de ambas técnicas. Ello implica que el uso de tecnologías de información tiene un papel muy significativo, ya que al usar formas de riego disruptivos se necesitan de herramientas innovadoras, y la tecnología puede ser una excelente aliada.

2.3. Sistema de riego manual

Sistema de riego realizado por personal humano que requiere de tiempo y recursos para ser operado. En un sistema de riego convencional se dispone de un circuito de tuberías donde cada cierto número de metros habrá una electroválvula que abastecerá el agua necesaria para el terreno. Normalmente, estas electroválvulas están controladas por una pareja de cables desde el controlador hasta la misma. Este par de cables debe recorrer la distancia que separe cada electroválvula del controlador, que en ocasiones será muy grande. Además, se utilizará un único par de cables por cada electroválvula, lo que supone un gasto muy grande de material. (Hernández, 2019)

3. COMPONENTES DEL SISTEMA DE RIEGO

3.1. Microcontrolador ESP-8266

El autor Artero lo define como: “Una placa hardware libre que incorpora un microcontrolador reprogramable y una serie de pines-hembra (los cuales están unidos internamente a las patillas de E/S del microcontrolador) que permiten conectar allí de forma muy sencilla y cómoda diferentes sensores y actuadores” (Artero, Ó. T., 2013).

3.2. Broker MQTT

El protocolo de transporte de telemetría de cola de mensajes o MQTT por sus siglas en inglés, fue ideado por IBM con el propósito de brindar una comunicación M2M dentro de infraestructuras IoT ya que este requiere poco ancho de banda para su funcionamiento dentro de dispositivos con poca CPU y RAM. Es ideal para sensores y microprocesadores.

MQTT fue creado por el Dr. Andy Stanford-Clark de IBM y Arlen Nipper de Arcom, ahora Eurotech, en 1999 como una forma rentable y confiable de conectar los dispositivos de monitoreo utilizados en las industrias del petróleo y el gas a servidores empresariales remotos. Este protocolo está basado en la topología estrella, la cual posee como nodo central un bróker que es el encargado de gestionar la red de conexiones con los dispositivos y transmitir los mensajes.

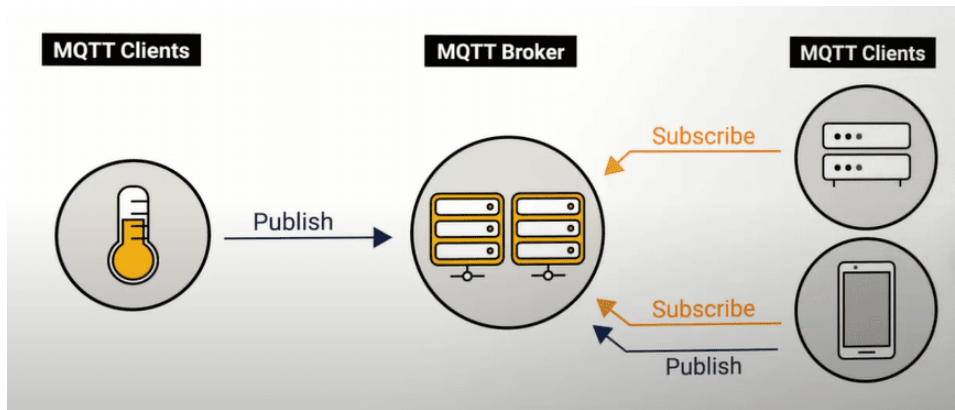


Ilustración 2 Estructura del MQTT

- a. MQTT – MENSAJES: Los mensajes son la información que se desea intercambiar entre los dispositivos. Puede ser un mensaje un comando o datos de lecturas de sensores, por ejemplo.
- b. MQTT – TÓPICOS: Otro concepto importante son los tópicos. Los tópicos son la forma en que registra interés por los mensajes entrantes o cómo especifica dónde desea publicar el mensaje. Los tópicos se representan con cadenas de texto separadas por una barra inclinada (/). Cada barra inclinada indica un nivel del tópico.

3.3. Eclipse Mosquitto

Es un bróker MQTT de código abierto que permite la implementación del protocolo MQTT (Figura 3). Además, es un intermediario de mensajes liviano de fácil uso para dispositivos de baja potencia como Arduino, o Raspberry pi.



Ilustración 3 Logo del Mosquitto

Características:







- Admite MQTTv3.1, v3.1.1 y v5.0
- Utiliza pocos recursos aptos para sensores de baja potencia, dispositivos móviles, y microcontroladores.
- Posee un repositorio de vulnerabilidades detectadas a tomar en consideración
- De fácil uso e implementación disponible para multiplataformas.

3.4. Node-Red

Ferencz, K., & Domokos, J. lo definen de la siguiente manera: “Node-RED es un entorno de desarrollo basado en JavaScript de código abierto basado en Node.js, desarrollado por ingenieros de IBM y más adecuado para desarrollar IoT sistemas. Más específicamente, es un entorno de programación virtual basado en procesos.” (Artero, Ó. T., 2019).

El Node-RED se caracteriza por poder crear nodos e instalarlos de manera sencilla, adaptándolos según las propias necesidades de la empresa o persona. A continuación, veremos algunos de los nodos con sus funciones que presenta este entorno:

Tabla 1 Listado de nodos del Node-RED

Nodo	Nombre	Descripción
	mqtt in	Se conecta a un agente MQTT y se suscribe a mensajes del tema especificado.
	Join	Une secuencias de mensajes a un solo mensaje.
	Debug	Muestra propiedades del mensaje a lado de la barra de depuración.
	Delay	Retrasa el mensaje que pasa a través del nodo o limita la velocidad del mensaje.
	Function	Bloque de funciones de JavaScript para ejecutarse con los mensajes que recibe el nodo.
	Dashboard	Permite crear dashboard o diseños.

3.5. XAMPP - Apache

Definimos el término de Xampp como “paquete de instalación de software libre que consiste en un sistema de gestión de base de datos MySQL y de servidor Apache, así como de intérprete de lenguajes PHP y Perl.” (Código B., 2017). El acrónimo de este término se refiere al uso de varios lenguajes como **A**pache, **M**ySQL, **P**HP y **P**erl.



Ilustración 4 Logo del Xampp

3.6. PhpMyAdmin

PhpMyadmin es un software libre que sirve para realizar gestiones de base de datos, permite además ejecutar consultas entre otras funciones, según se define en la página Docs (s.f.) como “herramienta de software libre escrita en PHP que está destinada a manejar la administración de un servidor de base de datos MySQL o MariaDB.”



Ilustración 5 Logo del phpMyAdmin

3.7. Componentes del sistema de riego

Para la realización del proyecto e implementación del sistema de riego se han utilizado algunos componentes, que listaremos a continuación:

Tabla 2 Listado de componentes utilizados

Componentes	Descripción
ESP8266	Es un microcontrolador con capacidad de conexión Wi-Fi, que no requiere un módulo externo para conectarse a redes inalámbricas.
Sensor DHT22	Sensor que permite la medición simultanea de temperatura y humedad
Sensor humedad del suelo	Sensor que permite la medición de la humedad

Cables para protoboard: hembra-hembra, macho - macho y hembra - macho	Se utilizan para el paso de la corriente y para hacer conexiones eléctricas, vemos los diferentes modelos con entradas variadas como son el macho y hembra.
Mini electrobomba sumergible de 5v	Se utiliza para bombear agua de un lugar a otro y pueden transportar agua limpia, sucia y líquidos varios, dependiendo del ramo empresarial o doméstico en el que se utilicen.
Relé: Módulo relay 1 canal 5 Vdc 10a	Interruptor eléctrico que permite dejar pasar y para la corriente eléctrica.

4. IMPLEMENTACIÓN DEL SISTEMA

4.1. Requisitos del sistema

4.1.1. *Requisitos funcionales*

Los requisitos funcionales son referidos a los servicios o funciones que proporciona el presente sistema de riego. Adicionalmente, los requisitos funcionales detallan las entradas, salidas, excepciones, entre otros aspectos.

- El sistema debe proveer proporcionar un servicio de riego controlado y monitoreado destinado al cultivo de la dracaena.
- El sistema debe permitir al usuario gestionar la lógica y datos a monitorear para realizar las pruebas de riego.
- El sistema debe capturar los datos de los sensores de humedad y temperatura.
- El sistema debe retornar una serie de resultados correspondiente a la temperatura en °C, temperatura del ambiente, humedad porcentual y humedad del suelo.
- El sistema debe mostrar la fecha y hora de los resultados obtenidos.
- El sistema debe proporcionar una conexión a una base de datos para recopilar los múltiples resultados de las pruebas realizadas.
- El sistema debe proporcionar una visualización de los resultados en un Dashboard.

4.1.2. Requisitos no funcionales

- El sistema de monitoreo debe visualizarse y funcionar correctamente en cualquier dispositivo de escritorio (computadora, laptop) o móvil (celular, tablet).
- El sistema funciona sobre una red local.

4.2. Arquitectura

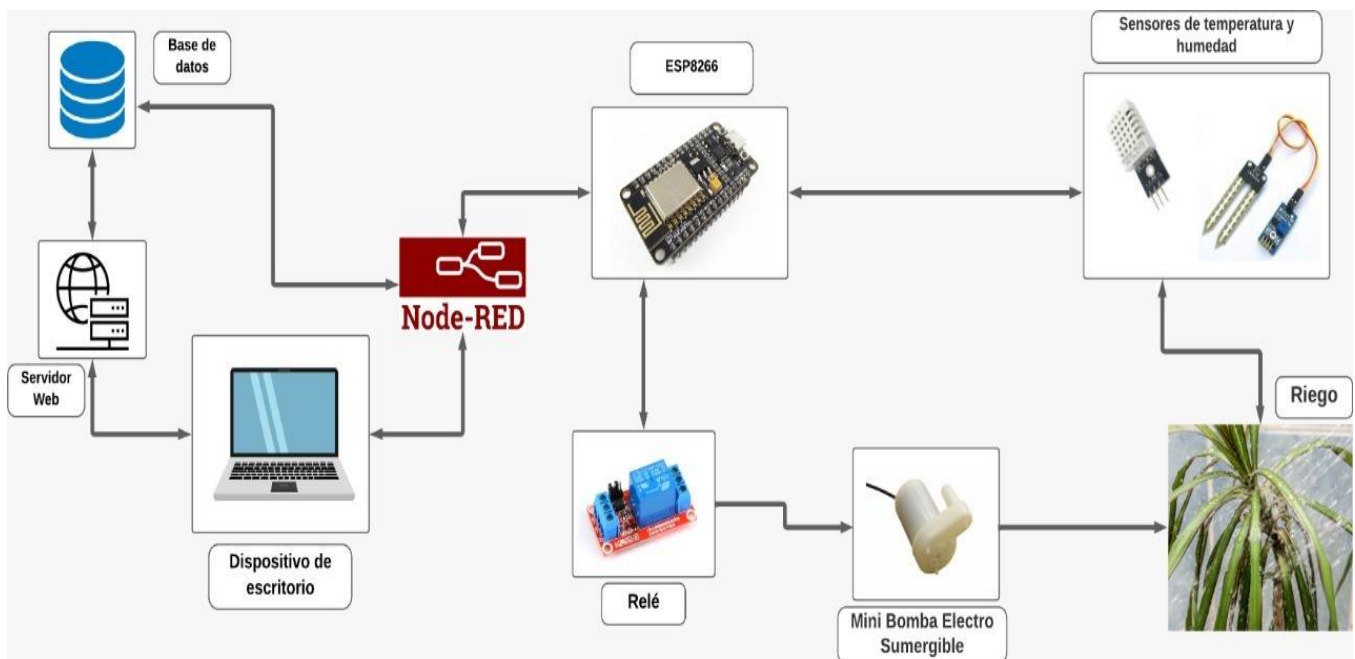


Ilustración 6 Arquitectura del sistema de riego de la dracanea

4.3. Diagrama Esquemático del Circuito de Riego en Wokwi

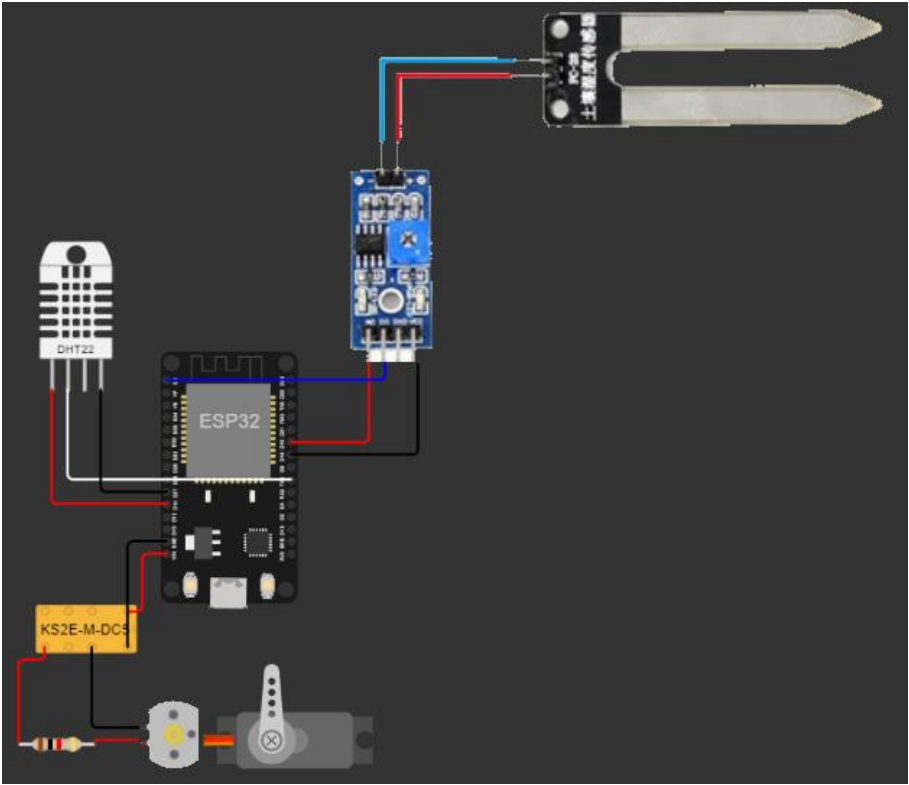


Ilustración 7 Arquitectura del sistema de riego de la dracanea

4.4. Diagrama de clase

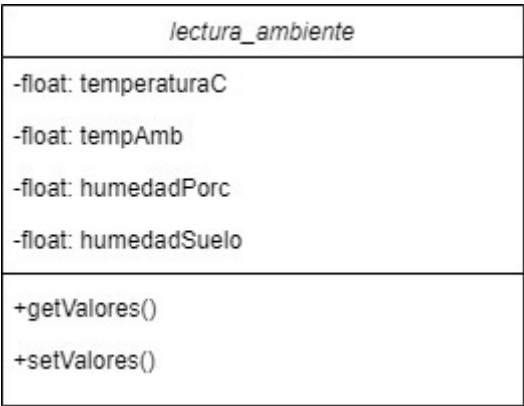


Ilustración 8 Estructura del diagrama de clase

4.5. Metodología del sistema

En este apartado explicaremos el proceso de implementación del sistema de riego desde el armado del circuito, la conexión con el MQTT server, el flujo de desarrollo con

el Node - RED, la conexión con la base de datos y la demostración del funcionamiento del sistema de riego.

Iniciamos probando el funcionamiento de la placa ESP8266 y de ambos sensores que serán utilizados en el sistema de riego, estos sensores son el DHT22 y el sensor de humedad. (véase las figuras 8, 9 y 10).

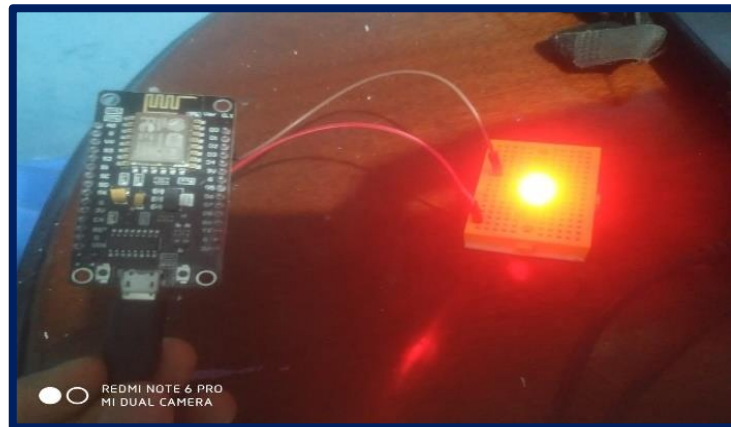


Ilustración 9 Funcionamiento de la placa

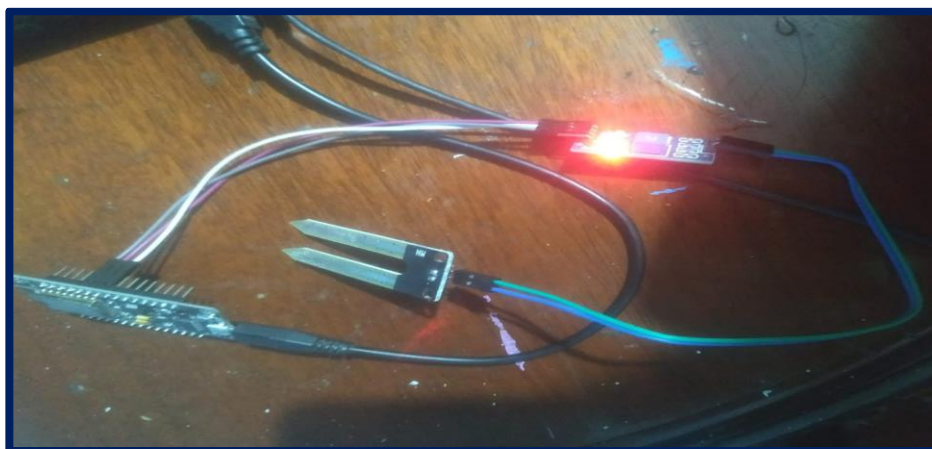


Ilustración 10 Funcionamiento del sensor de la humedad

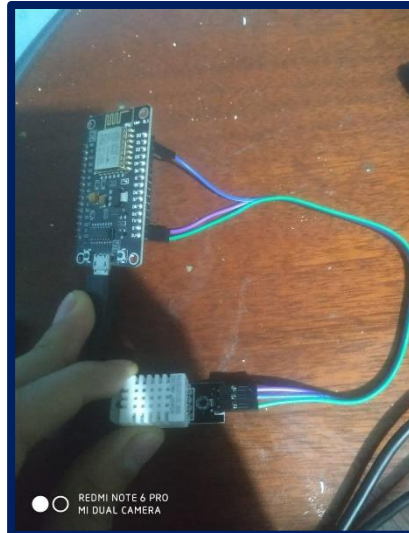


Ilustración 11 Funcionamiento del sensor DHT22

Luego de haber comprobado el buen funcionamiento de los tres dispositivos, pasamos al proceso de implementación, configuración y funcionamiento del protocolo de transporte conocido como el BROKER MQTT o Mosquitto (véase las figuras P y Q). Este protocolo nos permitirá poder conectar a través de un server a nuestro equipo de trabajo de manera remota.

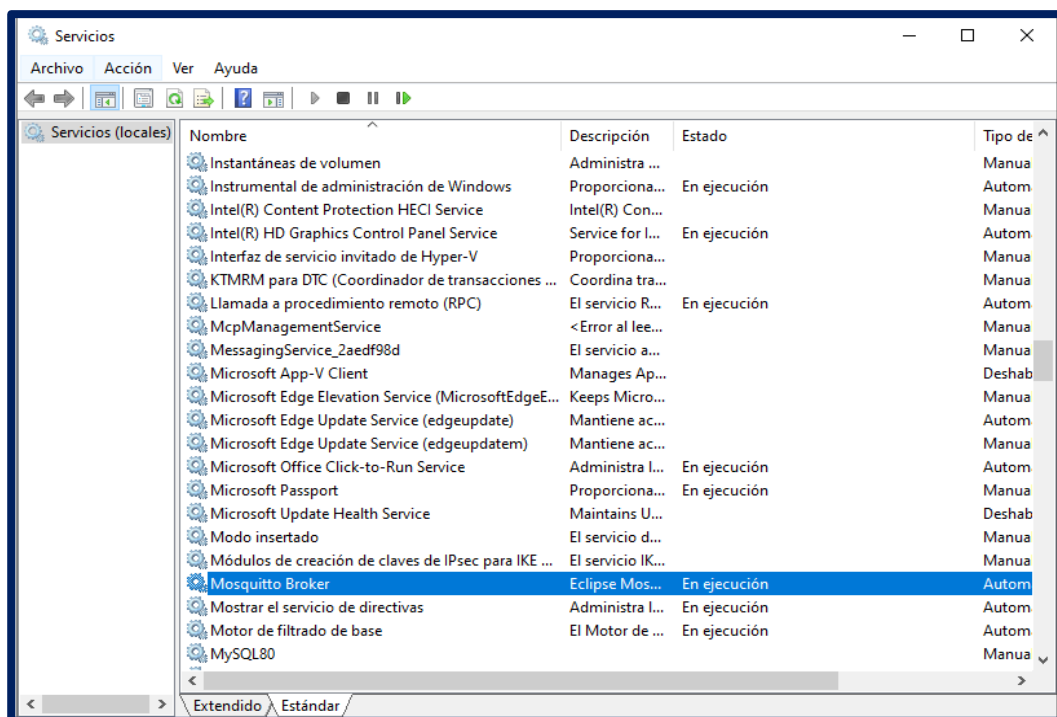


Ilustración 12 Implementación y funcionamiento del Broker MQTT en Windows

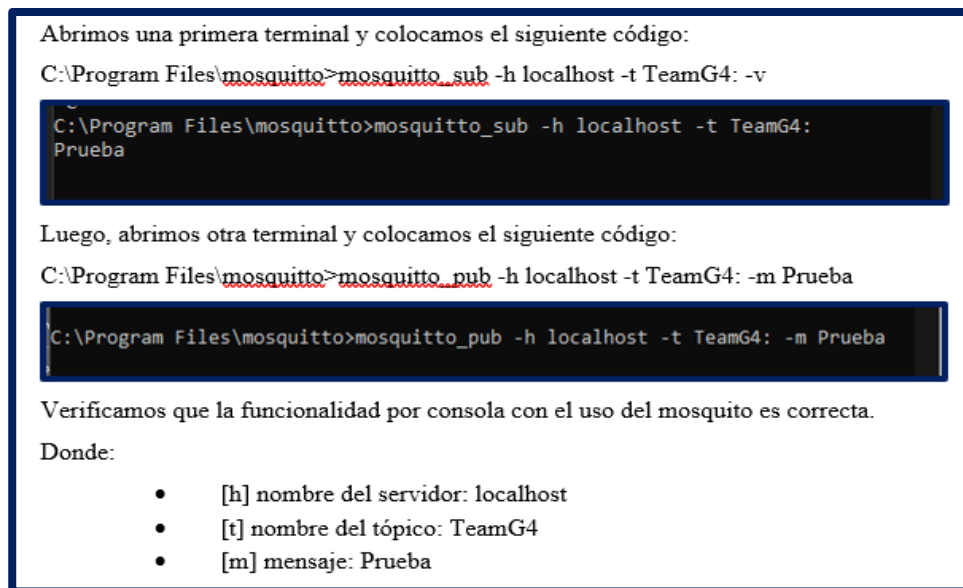


Ilustración 13 Funcionamiento del Broker MQTT por consola

Después de haber implementado de manera correcta el BROKER MQTT pasamos a la instalación y configuración del Arduino, instalando las librerías y tarjetas necesarias para el funcionamiento correcto de la placa ESP8266 y de los sensores, además el Arduino es nuestra base para poder armar el código y ver el funcionamiento del sistema de riego.

A continuación, haremos el armado del circuito, cabe resaltar que ya se ha ido armando el circuito durante las pruebas, lo que haremos es incorporar los nuevos componentes utilizados como la electrobomba de agua y el relé.

Después de haber armado el circuito, haremos las configuraciones necesarias para ver los resultados de los sensores a través de un dashboard, la conexión con el MQTT, la conexión con la base de datos, el almacenamiento de los datos de temperatura y humedad en tiempo real en la base de datos creada, todos estos procesos se explicarán mejor a continuación con el uso del entorno de desarrollo Node-RED, que servirá como nuestra base y es más accesible por la variedad de uso que nos presenta.

A continuación, explicaremos los pasos de la implementación en el Node-Red:

- a. Crear los nodos MQTT de entrada correspondientes al t3pico esp32/nombredesensor para suscribir los mensajes de lectura que marca cada sensor.

The screenshot shows the 'Edit mqtt in node' dialog box. Red arrows and text provide instructions:

- An arrow points to the 'Server' field (node-client@192.168.1.157:1883) with the text: **Client connection - client name + server address + port**.
- An arrow points to the 'Server' dropdown menu with the text: **select**.
- An arrow points to the 'Server' field with the text: **Edit client connection properties.**
- An arrow points to the 'Topic' field (sensors/+/control/#) with the text: **Topic to subscribe to**.
- An arrow points to the 'QoS' field (0) with the text: **Subscribe QOS**.
- An arrow points to the 'Name' field with the text: **Optional name that appears in admin pane. Uses topic if left blank**.

Ilustraci3n 14 Estructura del node mqtt

- b. Configurar los widgets chart con un identificador de lectura: Temperatura, Humedad ambiente o Sensaci3n t3rmica, y enlazarlo con su respectivo MQTT in (verificar el estado connected). En el caso del esp32/porchum se configura un nodo Gauge que marca el porcentaje de humedad que lee el sistema de riego.

The image shows two side-by-side screenshots of widget configuration dialog boxes:

- Left: 'Edit chart node' dialog box.** It shows settings for a chart widget, including:
 - Group: [Home] Group
 - Size: 6 x 4
 - Label: optional chart title
 - Type: Line chart
 - X-axis: last 1 hours OR 1000 points
 - X-axis Label: HH:mm:ss
 - Y-axis: min, max
 - Legend: None, Interpolate: linear
 - Series Colours: A set of color swatches.
 - Blank label: display this text before valid data arrives
 - Name: Name
- Right: 'Edit gauge node' dialog box.** It shows settings for a gauge widget, including:
 - Group: [Air Quality] Line grafic
 - Size: auto
 - Type: Gauge
 - Label: gauge
 - Value format: {{value}}
 - Units: units
 - Range: min 0, max 10
 - Colour gradient: A set of color swatches.
 - Sectors: 0, optional, optional, 10
 - Class: Optional CSS class name(s) for widget
 - Name: Name

Ilustración 15 Estructura del node chart

- c. Enlazar los nodos MQTT in al nodo Join permitirá unificar las secuencias de mensajes de cada MQTT por separado a uno solo.

Edit join node

Delete Cancel Done

Properties

Mode: manual

Combine each: msg. payload

to create: a key/value Object

using the value of: msg. topic as the key

Send the message:

- After a number of message parts: 4
- ☐ and every subsequent message.
- After a timeout following the first message: 10
- After a message with the msg.complete property set

Name: Name

☐ Enabled

Ilustración 16 Estructura del node join

- d. El nodo debug 1 muestra las propiedades del mensaje que envía el nodo Join en la pestaña de depuración. De forma predeterminada muestra msg.payload, pero se puede configurar para mostrar cualquier propiedad, el mensaje completo o el resultado de una expresión JSON. Por otro lado, el nodo limit 1msg/5min limita la velocidad del registro de mensajes a 1 mensaje por cada 5 minutos.

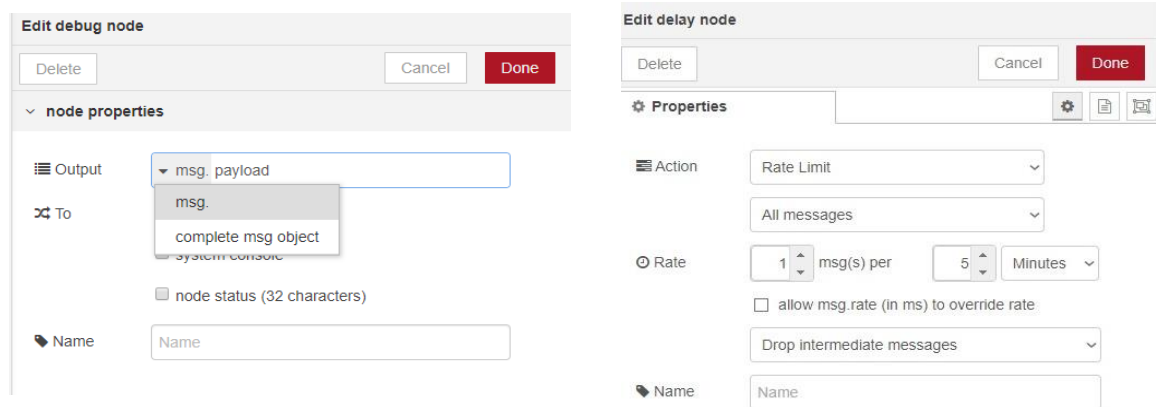


Ilustración 17 Estructura del node debug

- e. El nodo function permite leer un archivo JSON, el cual es traído desde el nodo Join, y permite crear una sentencia de SQL y se conecta con el nodo de base de datos.

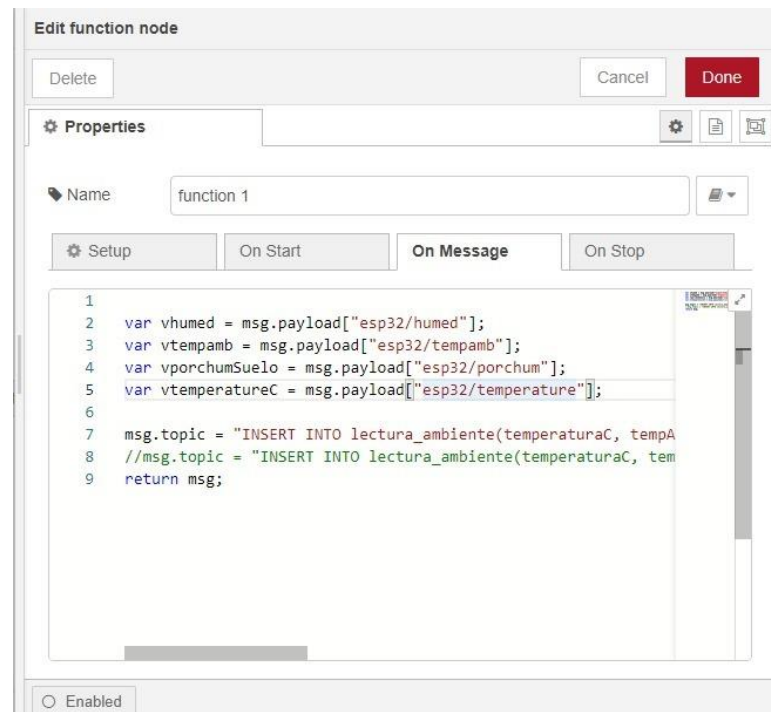


Ilustración 18 Estructura del node function

- f. Para la conexión de Node-red a la base de datos *iot-g4* creada en phpMyAdmin se utilizó el nodo MySQL que permite leer, escribir y hacer consultas en la base de datos MySQL. Para ello requiere el host, puerto, nombre de la base de datos, usuario y contraseña. Además, se pueden configurar otros parámetros como se muestra a continuación:

The image shows the configuration window for the MySQL node in Node-RED. The title bar reads "Edit mysql node > Edit MySQLdatabase node". At the top, there are three buttons: "Delete", "Cancel", and "Update". Below this is a "Properties" section with a gear icon and a document icon. The configuration fields are as follows:

- Host: 127.0.0.1
- Port: 3306
- User: root
- Password: (masked with dots)
- Database: iot-g4
- Timezone: ±hh:mm
- Charset: UTF8
- Name: Name

Below the fields is a yellow tip box that says: "Tip: The timezone should be specified as ±hh:mm or leave blank for". At the bottom, there is a status bar with "Enabled", "1 node uses this config", and a dropdown menu set to "On all flows".

Ilustración 19 Estructura del node mysql

4.3.2.3. Capturas de lo implementado en Node-RED

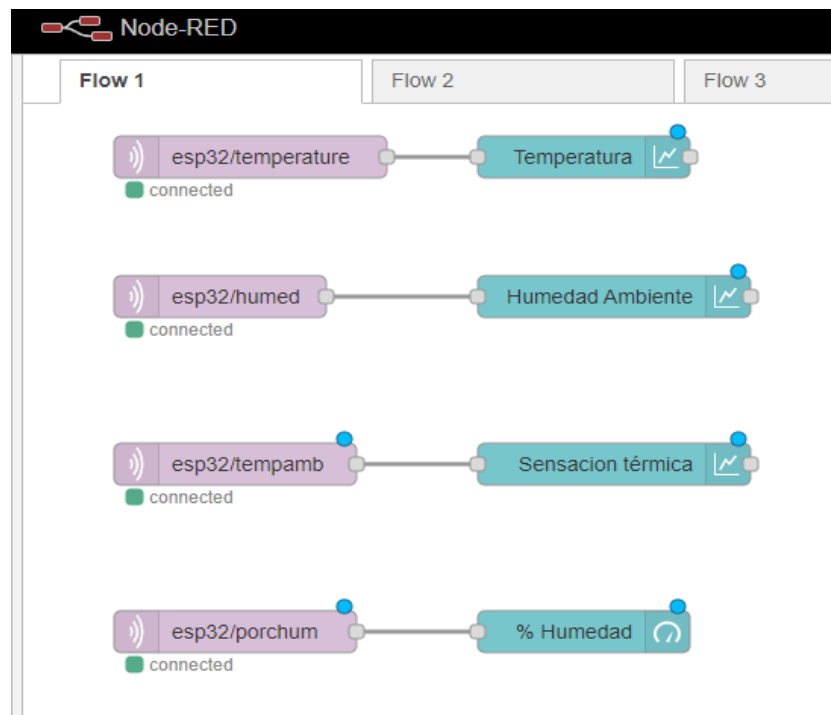


Ilustración 20 Estructura de los nodos de dashboard en el Node-Red

En esta imagen se puede ver la creación de 4 nodos de tipo MQTT en los cuales tienen configurado cada uno los tópicos que se ingresaran en el código para su recepción de mensaje. Luego, se agrega a cada uno los nodos dashboard donde se configura con un nombre y agregándolo a un respectivo grupo para que el grafico se muestre.

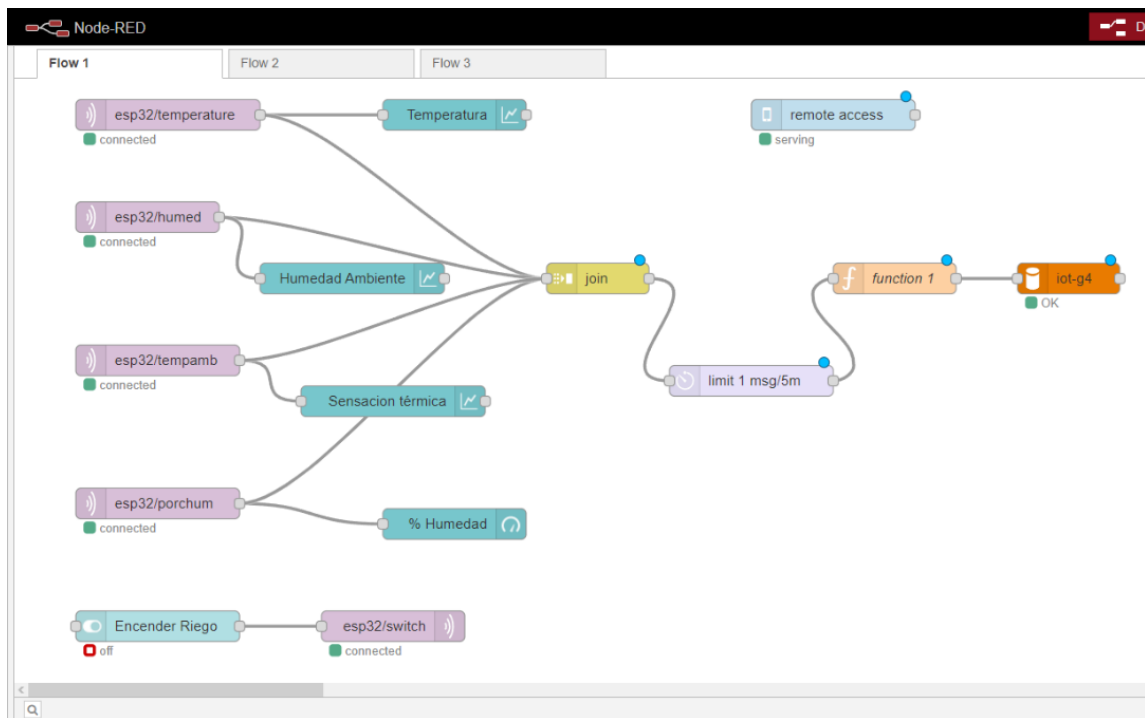


Ilustración 21 Estructura del sistema de riego en el Node-Red

En la siguiente imagen se puede ver como el nodo join une los mensajes de los nodos mencionados en la imagen anterior para así poder crear un objeto .json de tipo clave valor. El nodo join se configuró con modo manual y se define su retorno a un objeto de tipo json. Dicho de otro modo, el nodo join junta los nodos anteriores en un archivo de tipo json.

También se le añadió el nodo que limita el flujo para poder enviar cada 5 minutos un mensaje hacia la base de datos, con el objetivo de no tener registros cargados cada segundo de datos en la base de datos.

El nodo que limita el tiempo se conecta con el nodo function, el cual crea la query con los valores del json generado en el nodo join y la retorna. Finalmente, este nodo function se conecta al nodo de base de datos, con lo que se finaliza.

En la siguiente imagen se puede ver los 4 grupos que muestran las imágenes de los nodos dashboard que se vio en las imágenes anteriores. Se les conecto con los nodos principales para poder visualizar los datos en tiempo real.

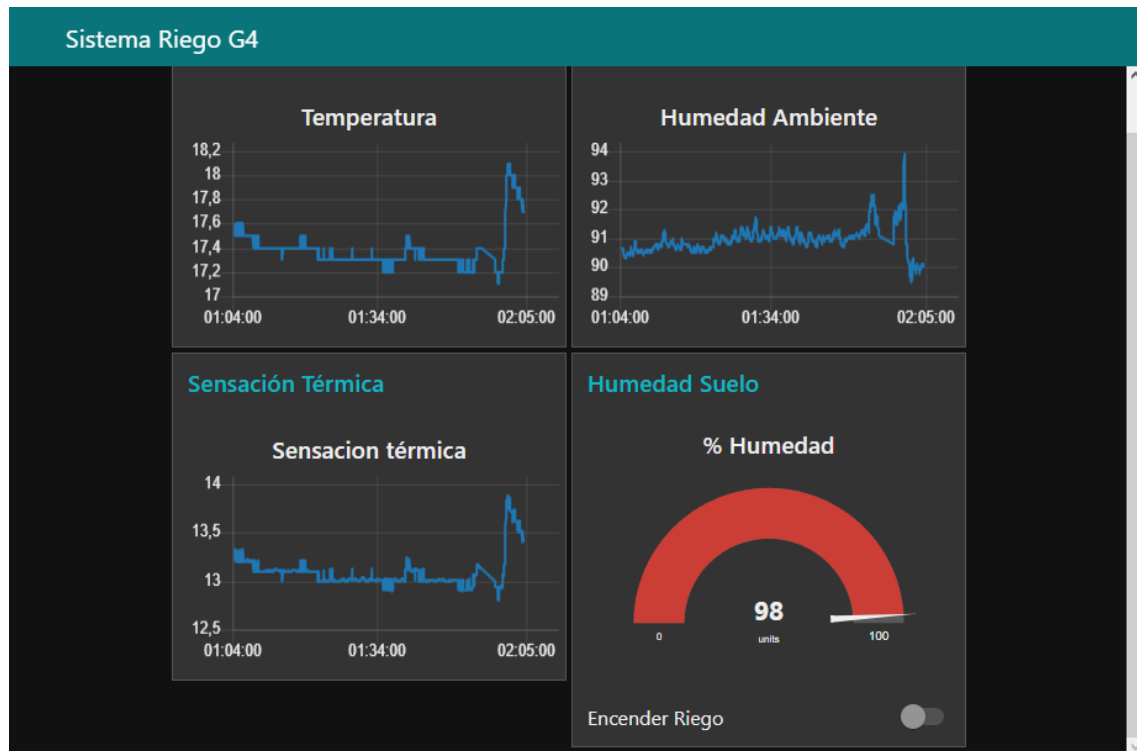


Ilustración 22 Dashboard de los sensores en tiempo real

En la siguiente imagen se puede ver los 4 grupos donde serán demostrados los nodos dashboard, hay un grupo por cada nodo.

En la siguiente imagen me muestra los 4 grupos donde cada una se encarga de mostrar la temperatura, humedad ambiente, la humedad del suelo y sensación térmica. Estos grupos mostraran los gráficos de los nodos dashboard que se vieron antes.

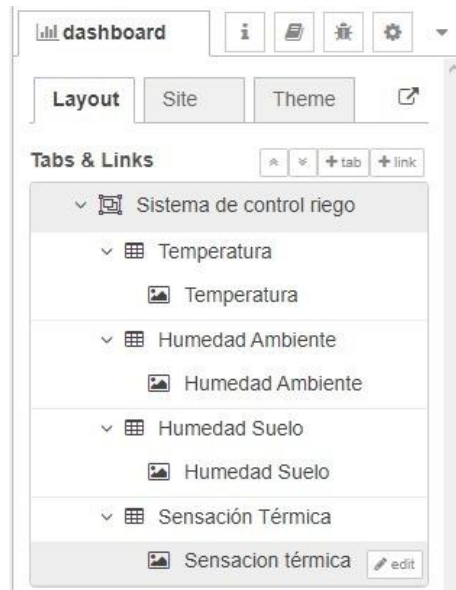


Ilustración 23 Estructura del dashboard en el Node-Red

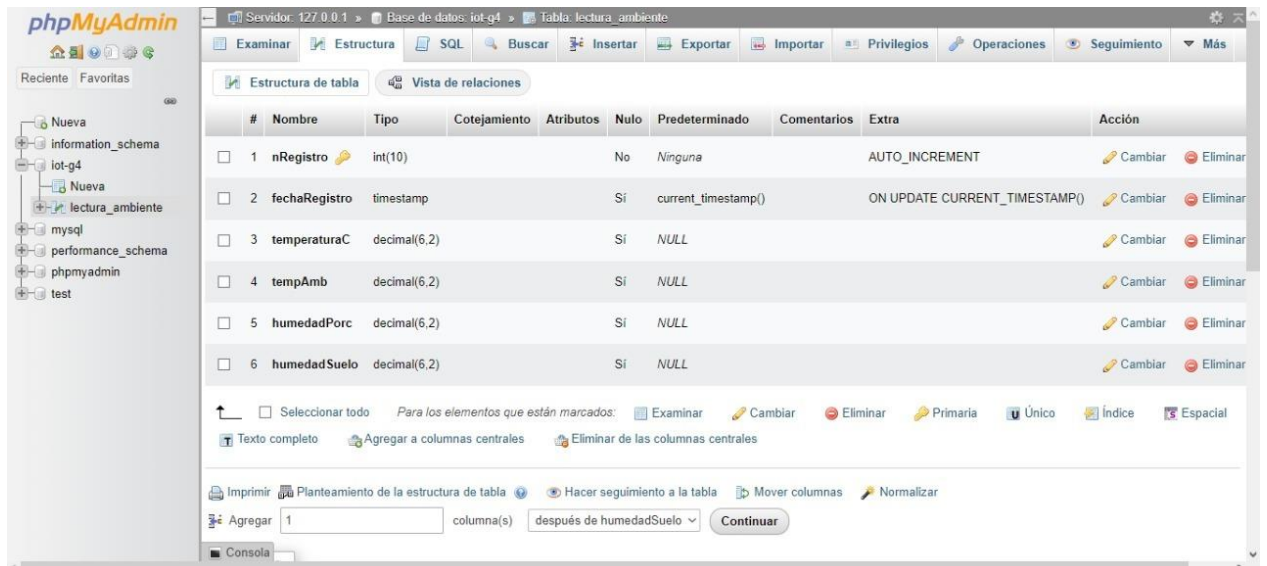


Ilustración 24 Captura de las columnas en la base de datos en el PhpMyAdmin

En la siguiente imagen se puede ver los campos de la tabla lectura_ambiente. Esta tabla registrara la fecha de registro, temperatura en Celsius, temperatura ambiente, temperatura en forma de porcentaje y la humedad de suelo.

phpMyAdmin

Reciente Favoritas

Base de datos: **iot-g4** » Tabla: **lectura_ambiente**

Examinar Estructura SQL Buscar Insertar Exportar Importar Más

☐ Perfilando [[Editar en línea](#)] [[Editar](#)] [[Explicar SQL](#)] [[Crear código PHP](#)] [[Actualizar](#)]

1 > >> | Número de filas: 25 | Filtrar filas: Ordenar según la clave: Ninguna

Opciones extra

				nRegistro	temperaturaC	tempAmb	humedadPorc	humedadSuelo	hora	fecha
<input type="checkbox"/>	Editar	Copiar	Borrar	1	21.00	16.00	71.00	100.00	21:21:52	2022-08-22
<input type="checkbox"/>	Editar	Copiar	Borrar	2	20.00	15.00	73.00	100.00	21:22:52	2022-08-22
<input type="checkbox"/>	Editar	Copiar	Borrar	3	19.00	14.00	74.00	100.00	21:23:52	2022-08-22
<input type="checkbox"/>	Editar	Copiar	Borrar	4	19.00	14.00	75.00	100.00	21:24:53	2022-08-22
<input type="checkbox"/>	Editar	Copiar	Borrar	5	19.00	14.00	76.00	100.00	21:25:53	2022-08-22
<input type="checkbox"/>	Editar	Copiar	Borrar	6	19.00	14.00	76.00	100.00	21:26:53	2022-08-22
<input type="checkbox"/>	Editar	Copiar	Borrar	7	19.00	14.00	76.00	100.00	21:27:53	2022-08-22
<input type="checkbox"/>	Editar	Copiar	Borrar	8	18.00	14.00	77.00	100.00	21:28:53	2022-08-22
<input type="checkbox"/>	Editar	Copiar	Borrar	9	18.00	14.00	77.00	100.00	21:29:53	2022-08-22
<input type="checkbox"/>	Editar	Copiar	Borrar	10	18.00	14.00	78.00	100.00	21:30:54	2022-08-22
<input type="checkbox"/>	Editar	Copiar	Borrar	11	18.00	14.00	77.00	100.00	21:31:54	2022-08-22
<input type="checkbox"/>	Editar	Copiar	Borrar	12	18.00	14.00	77.00	100.00	21:32:54	2022-08-22

Consola

Ilustración 25 Captura de los registros en la base de datos

Y por último en esta imagen se puede ver los datos registrados en la tabla `lectura_ambiente`. Cada cinco minutos se insertará un registro nuevo gracias al nodo `limits` que se vio antes.

5. DESPLIEGUE DE APP DASHBOARD CON PYTHON-DASH

Tras recopilar múltiples resultados del sistema de riego destinado al cultivo de draconeá, los datos son exportados a un dataset de extensión csv con el fin de desplegar una aplicación web que contenga un dashboard o cuadro de mando donde el usuario pueda visualizar los resultados por cada día que se ejecutó el servicio. Cabe destacar que los resultados corresponden a las fechas comprendidas entre 22/08/2022 al 26/08/2022.

5.1. Herramientas utilizadas

- **Visual Studio Code**, es el IDE o entorno de desarrollo seleccionado para la elaboración de esta aplicación.
- **Miniconda**, similar al ecosistema de Anaconda, es una herramienta que facilita el uso de librerías y gestión de paquetes de python de un proyecto dentro de un entorno virtual.
- **Jupyter Notebook**, es una interfaz web de código abierto empleado en este proyecto para visualizar preliminarmente los diversos gráficos basados en los resultados de las pruebas del sistema de riego.
- **Python**, es el lenguaje de programación seleccionado debido a su gran variedad de herramientas y librerías para el análisis y visualización de datos, de igual manera, para el apartado de web y aplicaciones.
- **Dash**, este framework de Python es popularmente utilizado para la construcción de aplicaciones web, sobre todo para visualización de datos y dashboards.
- **Github**, este sitio de ‘social coding’ aloja el api generado a partir del dataset la cual es solicitada por la app del dashboard, igualmente aloja en el mencionado sitio.

- **Heroku**, se optó por esta plataforma para desplegar tanto nuestra api y app web.

5.2. Preparación de un entorno virtual

En el entorno de Visual Studio Code creamos el proyecto MQTTRESULTS con la siguiente jerarquía de carpetas con sus respectivos archivos como se aprecia en la figura a continuación.

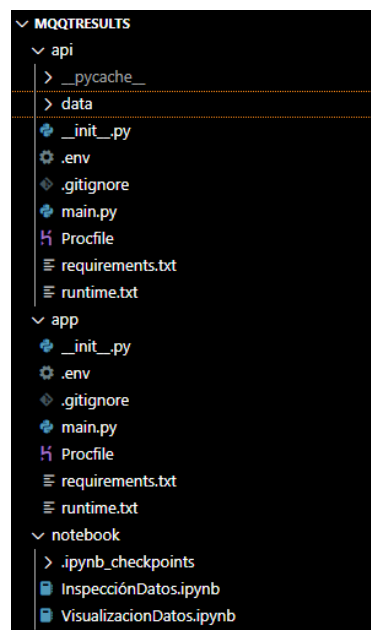


Ilustración 26 proyecto en Visual Studio Code

En la carpeta 'notebook' se crearon los archivos de extensión ipynb, correspondientes a un notebook de Jupiter. En el caso de 'InspecciónDatos.ipynb', se ha realizado un análisis de datos del dataset de resultados con el objetivo de encontrar alguna inconsistencia y realizar alguna operación necesaria como dividir en columnas separadas el valor de fecha y hora. Tras obtener el dataset revisado y limpio este se guardará en uno nuevo destinado para la visualización de este y consultas en el api.

	nRegistro	fechaRegistro	temperaturaC	tempAmb	humedadPorc	humedadSuelo	fecha	hora
0	1	2022-08-22 21:21:52	21.0	16.0	71.0	100.0	2022-08-22	21:21:52
1	2	2022-08-22 21:22:52	20.0	15.0	73.0	100.0	2022-08-22	21:22:52
2	3	2022-08-22 21:23:52	19.0	14.0	74.0	100.0	2022-08-22	21:23:52
3	4	2022-08-22 21:24:53	19.0	14.0	75.0	100.0	2022-08-22	21:24:53
4	5	2022-08-22 21:25:53	19.0	14.0	76.0	100.0	2022-08-22	21:25:53

Ilustración 27 Extracto del dataframe de los resultados revisados

En ‘VisualizaciónDatos.ipynb’ se realizaron preliminarmente los múltiples gráficos destinados al dashboard de la aplicación web del presente proyecto.

Por otro lado, las carpetas ‘api’ y ‘app’ comparten los nombres de una serie de archivos:

- .env, contiene las variables del entorno requeridas para su aplicación.
- main.py, corresponde al código del programa principal
- Procfile, este archivo señala cómo debe correr una aplicación de python, por ejemplo, como una simple aplicación web.
- requirements.txt, comprende todas las librerías o paquetes de python empleadas incluyendo sus versiones.
- runtime.txt, corresponde a la versión de python utilizada para el proyecto.

5.3 Generación de API

El propósito de generar un api es para que cumpla las solicitudes de la app web para retornar las gráficas, por ello, el api almacenará los datos obtenidos de las pruebas en formato json.

Para el desarrollo del api se optó tomar la fecha de los resultados como variable solicitante, de este modo, se retornará solo los datos de temperatura en °C, temperatura del ambiente, humedad porcentual y humedad del suelo que solo correspondan a la fecha solicitada.

5.4. Desarrollo del APP-Dashboard

Esta aplicación está compuesta en su mayoría por las sentencias creadas en el jupyter-notebook ‘VisualizaciónDatos.ipynb’. Además, es pertinente indicar que el archivo ‘.env’ de la app contiene una variable denominada ‘HOST_API’, la cual señala la dirección donde se aloja nuestro api, en un inicio, esta está alojada localmente para realizar las pruebas tempranas, más adelante, se cambia por un enlace proveído de un despliegue en Heroku.

La app web está comprendida principalmente por las siguientes funciones:

- `def variable_riego_info_dframe(fecha: str) -> pd.DataFrame`, responsable de obtener los datos del api y convertirlos en un dataframe para su posterior visualización.
- `def fig_dashboard_plots(df: pd.DataFrame) -> tuple`, se encarga de generar los gráficos de barras, distribución y serie de tiempo basados en los datos de temperatura en °C, temperatura del ambiente, humedad porcentual y la humedad del suelo.
- `def update_output_descrip(n_clicks, cvalue: str)`, actualiza la descripción de la fecha del dashboard correspondiente.
- `def update_output_plots(n_clicks, cvalue: str)`, actualiza los gráficos con respecto a la nueva fecha seleccionada.

También, gracias al framework de Dash, se recrea la variable ‘app’ que nos servirá de plantilla o estructura para la composición de la aplicación web, de este mod, cuenta con las siguientes sentencias:

- `app = Dash(__name__, external_stylesheets=[dbc.themes.MORPH])`, instanciación de la aplicación Dash considerando la estética proveída por la librería `dash_bootstrap_components`.
- `app.layout = html.Div(...)`, corresponde al diseño de la estructura de la aplicación web que comprende de header, una fila de botones, tres filas de gráficos y un footer.
- `@app.callback(...)`, gestiona las entradas y salidas dentro de la aplicación web.

Para una amplia vista al código relacionado a la aplicación web del proyecto, se invita visitar al siguiente enlace de GitHub: https://github.com/PietroUNMSM/IOTG04_app

5.5 Despliegue en Heroku

Para realizar el despliegue en Heroku es necesario contar con la configuración del entorno virtual señalado en puntos anteriores y haber subido tanto el desarrollo del api y la app web en repositorios de Github por separado.

5.5.1. Despliegue del api

Listo el repositorio del api, se procede con el despliegue de este en la plataforma de Heroku. Para ello basta con una cuenta gratuita, dirigirnos a ‘create a new app’, indicamos un nombre para él, luego, conectamos nuestro repositorio con la plataforma.

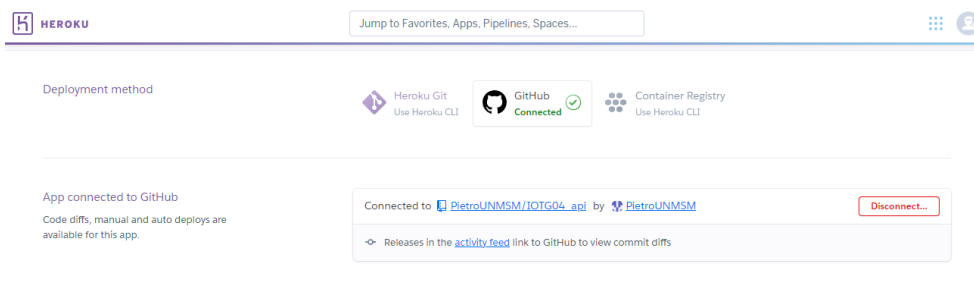


Ilustración 28 Conexión del repositorio del api en Heroku

Por último, nos dirigimos a la última opción ‘deploy’ para iniciar el despliegue y listo.

5.5.2. Despliegue de la app web

El protocolo para este despliegue es el mismo que el punto anterior.

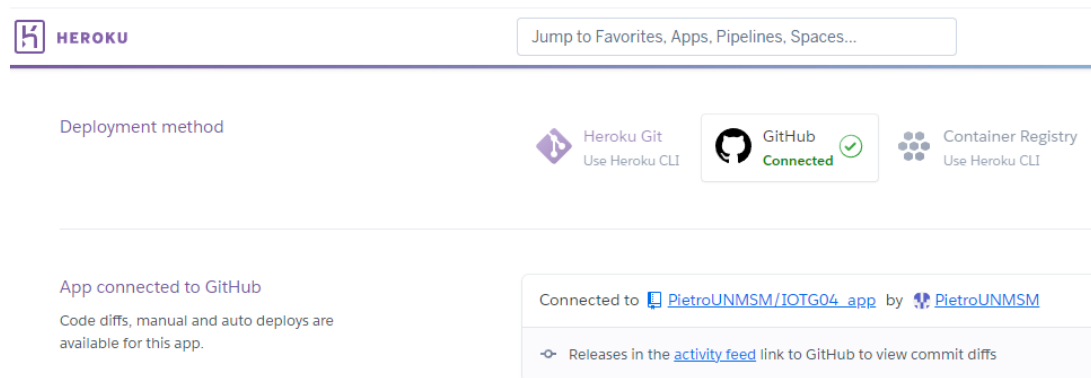


Ilustración 29 Conexión del repositorio de la app en Heroku

5.6. Resultado del despliegue

A continuación, se presenta la consulta de los datos recolectados del 24/08/2022, es visible a través del siguiente enlace del api desplegado:

<https://iotg04api.herokuapp.com/fecha/2022-08-24>

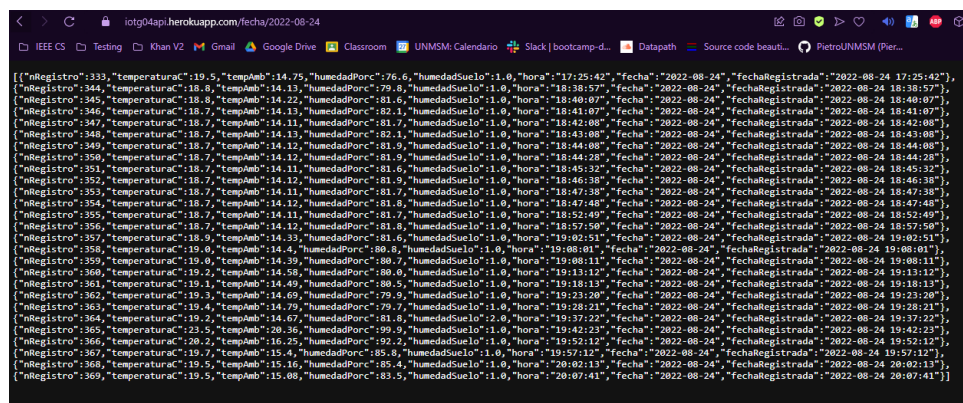


Ilustración 30 Resultados de api desplegado

Además, se presenta la visualización de los resultados recolectados del 26/08/2022 mediante la aplicación web del dashboard del proyecto.

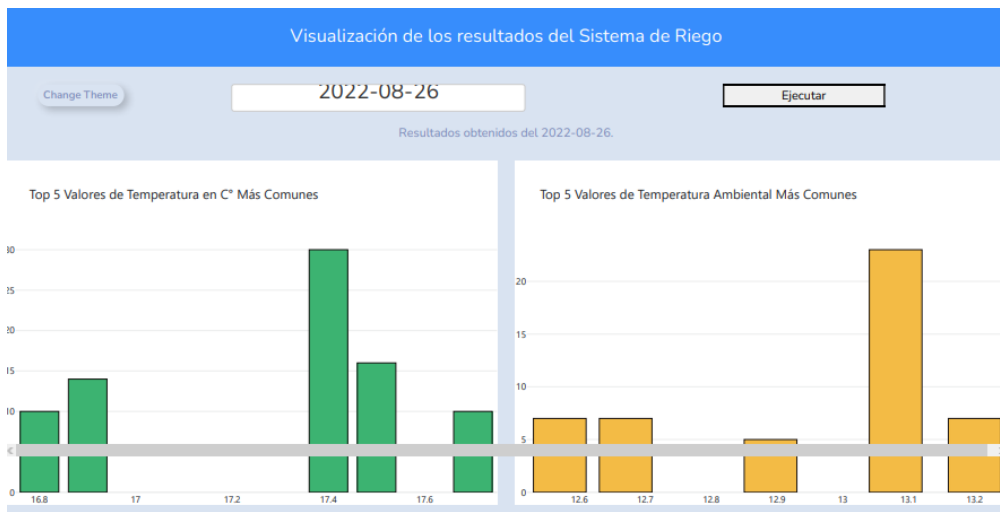


Ilustración 31 Valores de temperatura del 26/08/2022

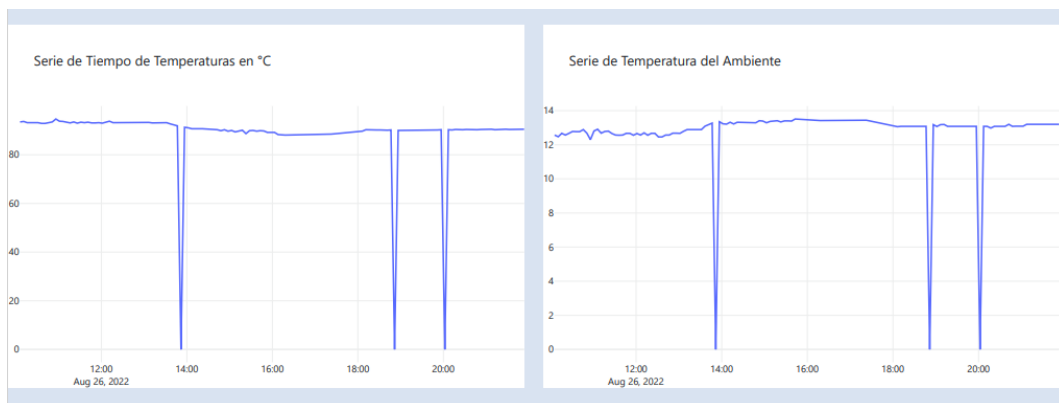


Ilustración 32 Series de tiempo del 26/08/2022

Adjuntamos los enlaces para visitar los despliegues:

- API: <https://iotg04api.herokuapp.com>
- APP/DASHBOARD: <https://iotg04dashboardapp.herokuapp.com>

6. RESULTADOS

Para poder discutir y mostrar los resultados es importante detallar lo que el proyecto ha podido generar. En base a la medición de la temperatura y la humedad y con ello lograr desarrollar un sistema de riego se ha generado un historial de datos que hemos organizado gráficamente. Este historial de datos ha sido desplegado en un aplicativo web y está almacenado dentro de la plataforma Heroku.

La visualización de los datos está organizada por fecha, dentro del día seleccionado tenemos varias gráficas que en base a las horas pueden mostrar:

- Top 5 valores de temperatura más comunes
- Top 5 valores de temperatura ambiental más comunes
- Top 5 valores de humedad porcentual más comunes
- Top 3 valores de humedad de suelo más comunes
- Distribución de los valores de temperatura
- Series de temperatura, temperatura ambiental, humedad porcentual y humedad del suelo en base al tiempo.

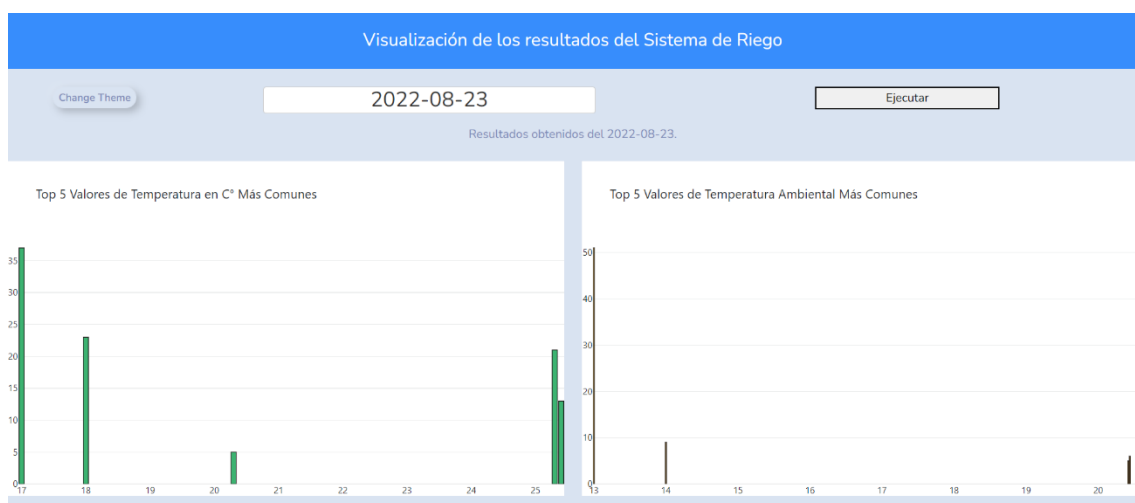


Ilustración 33 Portada general

Nota: Primeros dos gráficos del aplicativo web que muestra los datos históricos

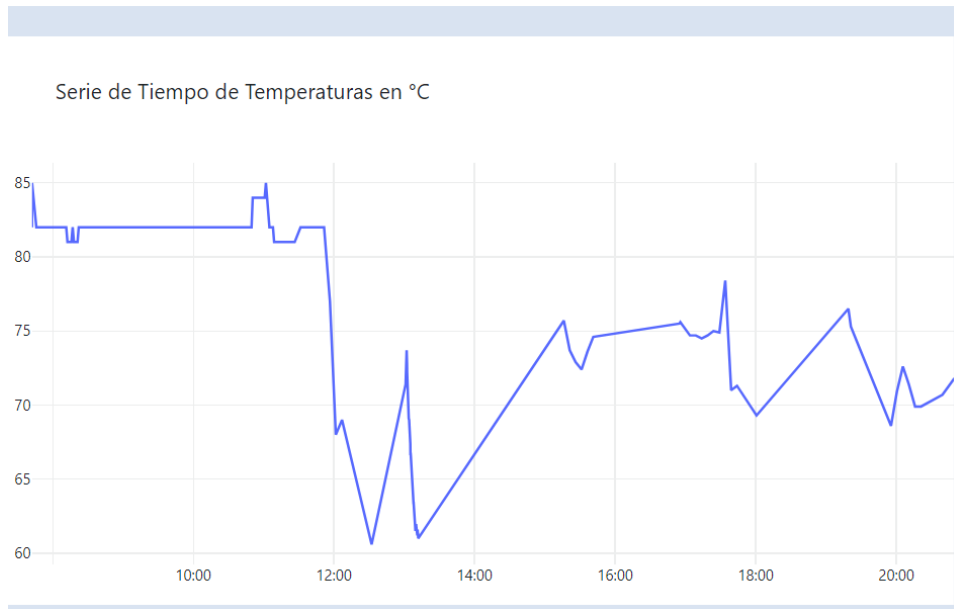


Ilustración 34 Serie de temperatura

Nota: Data de la temperatura en base al horario en que fue recogida la data



Ilustración 35 Serie de temperatura ambiental

Nota: Data de la temperatura del ambiente en base al horario en que fue recogida la data

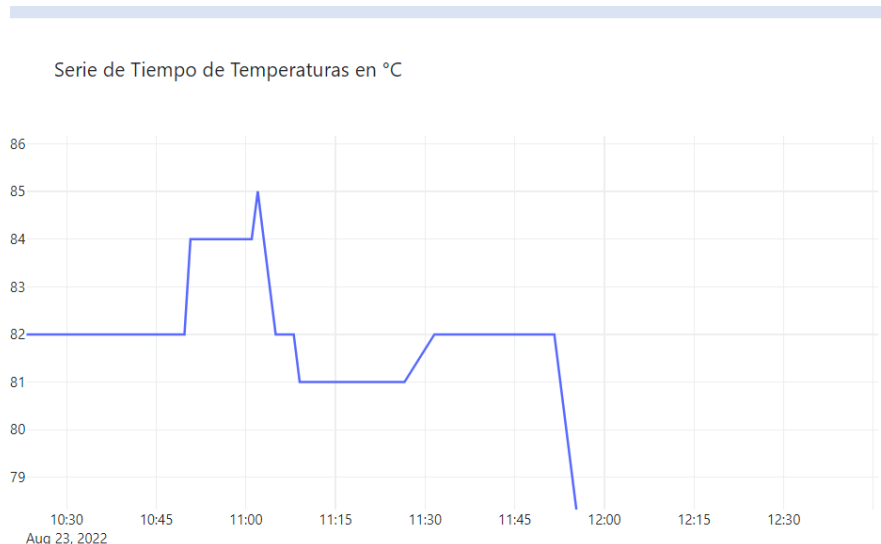


Ilustración 36 Acercamiento gráfico

Nota: Es posible realizar un acercamiento a los gráficos para así evidenciar la data de forma más precisa respecto al tiempo



Ilustración 37 Organización de los componentes y sensores del sistema de riego

Nota: Conexiones y organización de los componentes

De la imagen anterior es necesario mencionar que toda la estructura y conexiones fueron dadas con el objetivo de desarrollar un sistema de riego eficaz y que esté acorde a lo que los sensores puedan ordenar. Es sabido que gracias al conocimiento de la temperatura y humedad las plantas pueden ser mejor tratadas, y por ello fue necesario usar componentes electrónicos, algunos elementos de plástico como tubos, una planta de prueba y demás materiales como cables extra y el agua misma. Para que de esa forma se pueda llegar a dar un tratamiento eficiente de riego a una planta cuyos datos fueron mencionados anteriormente.

6.1 Discusión

De los gráficos mostrados anteriormente es posible mencionar que dentro de los días en que se tomó la data, la temperatura que más veces pudo darse fue de 17 grados Celsius, la temperatura de ambiente más común fue de 13 grados Celsius. También el pico más alto de temperatura fue de 86 grados y el pico más alto de temperatura de ambiente fue de 20.5 grados. Esta información nos ayuda a entender la dinámica de cómo los datos se presentan y podremos tomar mejores decisiones cuando observamos ciertos patrones.

Si la temperatura más común fue la de 17 grados, podremos usar esta información y tomar precauciones de cuidado hacia la planta o mejoras en cuándo activar o no el riego. Lo mismo para la temperatura de ambiente, si conocemos que el dato más común fue de 13 grados, con la ayuda del marco teórico conocido de la planta, podremos establecer mejores regímenes de riego para así no dañar su vitalidad. Cuando hallamos picos de temperatura nos damos cuenta si nuestro sistema está preparado para ello o si la planta

puede no ser dañada con dichos índices. Así si la temperatura no es la más idónea para que la planta pueda ser regada, es necesario tomar otras opciones para cumplir con el objetivo planteado.

CONCLUSIONES

Se logro realizar un prototipo de sistema de riego inteligente aplicando los conocimientos de internet de las cosas, integrando tecnologías como Arduino, Node-red, phpMyAdmin en el prototipo inteligente. En tal sentido el sistema IoT planteado logra mostrar en tiempo real y de manera grafica los datos de las variables de temperatura, sensación térmica, humedad ambiental y humedad del suelo, que van captando los sensores gracias al Node-red.

Haciendo uso de la base de datos, se recopilaron los registros ambientales de un periodo de tiempo de una semana, junto a librerías utilizando Python se creó una aplicación web para visualizar gráficos con los datos de las variables.

Aunque en el mercado se encuentren sistemas autónomos y automatizados, la gran mayoría de ellos son difíciles de acceder debido al elevado costo económico que presentan, por esto se hace necesario el desarrollo de sistemas como el propuesto para disminuir el costo y hacerlo accesible.

BIBLIOGRAFÍA

Aguilar Zavaleta, S. (2020). Diseño de una solución basada en el internet de las cosas (IoT) empleando Lorawan para el monitoreo de cultivos agrícolas en Perú.

https://alicia.concytec.gob.pe/vufind/Record/UTPD_c9d0107e2f564074c0f5f45fdac24af9

Artero, Ó. T. (2013). ARDUINO. Curso práctico de formación. RC libros.

https://books.google.es/books?hl=es&lr=&id=6cZhDmf7suQC&oi=fnd&pg=PR15&dq=ARDUINO.+Curso+pr%C3%A1ctico+de+formaci%C3%B3n.+RC+libros.&ots=A-exo-JwyM&sig=i_uvw-GkLBg7Rp-_Pl1dX7g1w-s#v=onepage&q=ARDUINO.%20Curso%20pr%C3%A1ctico%20de%20formaci%C3%B3n.%20RC%20libros.&f=false

Ferencz, K., & Domokos, J. (2019). Using Node-RED platform in an industrial environment. XXXV. Jubileumi Kandó Konferencia, Budapest, 52-63.

https://www.researchgate.net/profile/Katalin-Ferencz/publication/339596157_Using_Node-RED_platform_in_an_industrial_environment/links/5e5ab48c4585152ce8fc6a6c/Using-Node-RED-platform-in-an-industrial-environment.pdf

González Breña, V. (2017). Riego inteligente (Bachelor's thesis).

Hernández Rodríguez, C. (2019). Sistema de riego inteligente de bajo coste.

Introducción — documentación de phpMyAdmin - 5.3.0-dev. (s. f.). docs.phpmyadmin.net. <https://docs.phpmyadmin.net/es/latest/intro.html>

J. (2017, 21 septiembre). *Xampp. Herramienta indispensable para desarrollo web.*

CodigoBinario por Jorge Grau. <https://www.codigo-binario.es/xampp-herramienta-para-dev-web/>

Mercado Garcia, J. C. (2020). Sistema de riego autónomo de bajo costo para expansión de área agrícola en laderas de los Valles del Sur del Perú basado en IOT.

https://alicia.concytec.gob.pe/vufind/Record/UNAL_8dd1f03ed778e47a1b49a80136d00fba

Picand, Y. D. D. (s. f.). DRACAENA MARGINATA: definición de DRACAENA MARGINATA y sinónimos de DRACAENA MARGINATA (español). Sensagent – 2005–2015. Recuperado 23 de agosto de 2022, de <http://diccionario.sensagent.com/DRACAENA%20MARGINATA/es-es/>

Sun, J., Liu, J. N., Fan, B., Chen, X. N., Pang, D. R., Zheng, J., ... & Li, J. (2019). Phenolic constituents, pharmacological activities, quality control, and metabolism of *Dracaena* species: A review. *Journal of Ethnopharmacology*, 244, 112138.

Yalle Arce, R. D. (2021). Automatización y telecontrol del sistema de riego para las áreas verdes del templo Santos de los Últimos Días – Arequipa. https://alicia.concytec.gob.pe/vufind/Record/UNAS_53a60f9448b123d46d239b3dfb24bbce

ANEXOS

Código fuente del proyecto:

```
//===== LIBRERIAS =====  
//Librerías del sensor DHT22 (sensor de humedad y temperatura)  
#include <DHT.h> // Cargamos la librería DHT para interactuar con el sensor de temperatura y humedad  
#include <DHT_U.h> // Cargamos la libreria DHT_U (DHT_Unificado)  
  
//===== PLACA ESP8266 =====  
//Librerías para la conexión del WIFI  
#include <ESP8266WiFi.h> // Incluyendo ESP8266WiFi.h para conectar el modulo ESP8266 a la red  
WIFI  
#include <PubSubClient.h> // Esta biblioteca le permite enviar y recibir mensajes MQTT  
  
// ===== SENSOR DHT22 =====0  
DHT dht (D6, DHT22); // Definiendo los puertos para el sensor DHT.  
// temp tendra la tempreatura en celcius, temp_f tendra la temperatura en fahrenheit,  
// humed tendra la humedad y hi tendra la sensacion termica.  
float temp, temp_f, humed, hi;  
  
//===== VARIABLES DE RED =====  
//Conexión a la red WIFI  
const char* ssid = "bacco 2.4 GHZ"; //nombre de la red  
const char* password = "frecuencia34"; //contraseña de la red  
const char* mqtt_server = "192.168.0.15"; //ip del broker  
  
//Inicializamos el objeto de cliente esp  
WiFiClient espClient;  
  
//Iniciamos el objeto subscriptor del cliente con el objeto del cliente  
PubSubClient client(espClient);  
unsigned long lastMsg = 0; // Declaracion de variable de tipo long.  
#define MSG_BUFFER_SIZE (300) // Defininendo la longitud del buffer.  
char msg[MSG_BUFFER_SIZE]; // Creando un vector de caracteres de nombre msg.  
int value = 0; // Declaracion de variable value incializado en 0.  
  
//===== SETUP DEL WIFI =====  
void setup_wifi() { // Declaracion de la funcion setup_wifi  
  delay(10); // Paraliza la ejecucion pro 10 milisegundos  
  // Empezamos por conectarnos a una red WiFi
```

```

Serial.println(); // Se imprime en el monito en serie un salto de linea
Serial.print("Connecting to "); //Se imprime un mensaje de conectando
Serial.println(ssid); // Nos muestra el nombre de la red conectada
//WiFi.mode(WIFI_STA);
WiFi.begin(ssid, password); // Inicializa la configuración de red de la biblioteca WiFi
while (WiFi.status() != WL_CONNECTED) {
    delay(500); // Paraliza la ejecucion por medio segundo
    Serial.print("."); // Se imprime en el monitor en serie "."
}
//Proceso q nos menciona q el WIFI está conectado
randomSeed(micros()); // Inicializando el generador de números pseudoaleatorios iniciando el número
de microsegundos desde que la placa Arduino comenzó a ejecutar el programa actual.
Serial.println(""); // Se imprime en el monito en serie un salto de linea
Serial.println("WiFi connected"); // Se imprime en el monitor en serie "WiFi connected"
Serial.println("IP address: "); // Se imprime en el monitor en serie "IP address: "
Serial.println(WiFi.localIP()); // Se imprime en el monitor en serie la dirección IP del escudo WiFi.
} // Fin funcion

//==== FUNCION PARA RECIBIR MENSAJES DE NODE-RED USANDO MQTT ===
void callback(char* topic, byte* payload, unsigned int length) {
    Serial.print("Message arrived ["); // Se imprime en el monitor en serie "Message arrived ["
    Serial.print(topic); // Se imprime en el monitor en serie topic
    Serial.print("] "); // Se imprime en el monitor en serie "]"
    String messageTemp;
    for (int i = 0; i < length; i++) {
        Serial.print((char)payload[i]);
        messageTemp += (char)payload[i];
    }
    if (String(topic) == "esp32/switch") {
        if(messageTemp == "true"){
            Serial.println("on");
            digitalWrite(D8, HIGH);
        }
        else if(messageTemp == "false"){
            Serial.println("off");
            digitalWrite(D8, LOW);
        }
    }
}

//===== RECONEXION =====

```

```

void reconnect() {
    while (!client.connected()) { // Siempre que el cliente no este conectado
        Serial.print("Attempting MQTT connection...");
        // Se crea un cliente por default
        //creamos una nueva cadena de conexión para el servidor
        String clientId = "ESP8266Client-";
        clientId += String(random(0xffff), HEX);
        // Intento de conexión
        if (client.connect(clientId.c_str())) { // Si esta conectado
            Serial.println("connected"); // Se imprime en el monitor en serie "connected"
            // Una vez conectado, publica un anuncio...
            client.publish("outTopic", "hello world");
            // ... y volver a suscribirse
            client.subscribe("esp32/switch");
        } else { // Si no esta conectado
            Serial.print("failed, rc="); // Se imprime en el monitor en serie "failed, rc="
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            // Esperar 5 segundos antes de reintentar
            delay(5000);
        }
    }
}

void setup() {
    //===== PLACA ESP8266 =====
    pinMode(BUILTIN_LED, OUTPUT); // Inicializar el pin BUILTIN_LED como salida
    pinMode(D8, OUTPUT); // Establecer el pin D8 como salida
    pinMode(D7, INPUT); // Establecer el pin D7 como entrada
    pinMode(A0, INPUT); // Establecer el pin A0 como entrada
    Serial.begin(115200); // Estableciendo la velocidad de datos en 115200 bits por segundo
    setup_wifi(); // Numero del puerto
    client.setServer(mqtt_server, 1883); // Numero del puerto
    client.setCallback(callback); // Devolucion de llamada
    //===== SENSOR DHT22 =====
    dht.begin(); // Funcion usada para inicializar el sensor
    delay(500);
}

void loop() { // Funcion principal
    //===== SENSOR DHT22 =====
    humed = dht.readHumidity(); //Captando la humedad del sensor

```

```

temp = dht.readTemperature (); //Captando la temperatura en grados celcius
temp_f= dht.readTemperature (true); //Captando la temperatura en grados Farenheit
hi= dht.computeHeatIndex (temp, humed); // Captando la sensacion termica

//===================================================== SENSOR HUMEDAD DE SUELO =====
int humedad_lectura = analogRead(A0); // Variable que capta el valor de la humedad del suelo
//Variable que tendra el valor de la lectua en forma de porcentaje con la funcion map
int lecturaSueloPorc = map(humedad_lectura, 0, 1023, 100, 0);

//===================================================== PLACA ESP8266 =====0
//Preguntamos por el estado de la conexión
if (!client.connected()) {
    reconnect();
}
client.loop();
unsigned long now = millis();
delay(10);
if (now - lastMsg > 2000) {
    lastMsg = now;
    ++value;

    char x[8];
    //Convertimos la variable temperatura a string
    dtostrf(temp,7,3,x);
    //Publicamos el mensaje en el siguiente topico:
    client.publish("esp32/temperature",x);
    //Convertimos la variable temperatura a string
    dtostrf(humed,7,3,x);
    //Publicamos el mensaje en el siguiente topico:
    client.publish("esp32/humed", x);
    //Convertimos la variable temperatura a string
    dtostrf(hi,7,3,x);
    //Publicamos el mensaje en el siguiente topico:
    client.publish("esp32/tempamb", x);
    //Convertimos la variable temperatura a string
    dtostrf(lecturaSueloPorc,7,3,x);
    //Publicamos el mensaje en el siguiente topico:
    client.publish("esp32/porchum", x);
}
}

```

Código fuente del API

```
#main.py#=====
=====# common
import os
import json
# requirements
from dotenv import load_dotenv
from fastapi import FastAPI
import pandas as pd

# -----

load_dotenv('./.env')

app = FastAPI()

# => routers
@app.get('/')
async def mainIndex() -> ...:
    return main_index()

#variable_riego (temperatura, humedad)-> float64 || fecha -> object
@app.get('/fecha/{fecha}')
async def variable_riegoInfo(fecha: str) -> ...:
    return variable_riegoInfo(fecha)

# => functions
def main_index() -> dict:
    return {'message': 'use esta api para tu visualizacion'}

def variable_riegoInfo(fecha: str) -> dict:
    host_data = os.environ['HOST_DATA']

    df = pd.read_csv(f'{host_data}/lectura_revisadaV2.csv', sep=',', encoding='utf-8')
    subdf = df[df['fecha'] == fecha]
    return json.loads(subdf.to_json(orient='records', date_format='iso'))
```

Para una amplia vista al código relacionado al api del proyecto, se invita visitar al siguiente enlace de GitHub: https://github.com/PietroUNMSM/IOTG04_api

Estructura de los datos de la base de datos en phpMyAdmin:

```
-- phpMyAdmin SQL Dump
-- version 5.2.0
-- https://www.phpmyadmin.net/
--
-- Servidor: 127.0.0.1
-- Tiempo de generación: 29-08-2022 a las 11:17:08
-- Versión del servidor: 10.4.24-MariaDB
-- Versión de PHP: 7.4.29

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
START TRANSACTION;
SET time_zone = "+00:00";

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

--
-- Base de datos: `iot-g4`
--
--
-- Estructura de tabla para la tabla `lectura_ambiente`
--
CREATE TABLE `lectura_ambiente` (
  `nRegistro` int(10) NOT NULL,
  `temperaturaC` decimal(6,2) DEFAULT NULL,
  `tempAmb` decimal(6,2) DEFAULT NULL,
  `humedadPorc` decimal(6,2) DEFAULT NULL,
  `humedadSuelo` decimal(6,2) DEFAULT NULL,
  `hora` time DEFAULT curtime(),
  `fecha` date DEFAULT curdate()
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

--
```

```

-- Volcado de datos para la tabla `lectura_ambiente`
--

INSERT INTO `lectura_ambiente` (`nRegistro`, `temperaturaC`, `tempAmb`, `humedadPorc`,
`humedadSuelo`, `hora`, `fecha`) VALUES
(1, '21.00', '16.00', '71.00', '100.00', '21:21:52', '2022-08-22'),
(2, '20.00', '15.00', '73.00', '100.00', '21:22:52', '2022-08-22'),
(3, '19.00', '14.00', '74.00', '100.00', '21:23:52', '2022-08-22'),
(4, '19.00', '14.00', '75.00', '100.00', '21:24:53', '2022-08-22'),
(5, '19.00', '14.00', '76.00', '100.00', '21:25:53', '2022-08-22'),
(6, '19.00', '14.00', '76.00', '100.00', '21:26:53', '2022-08-22'),
(7, '19.00', '14.00', '76.00', '100.00', '21:27:53', '2022-08-22'),
(8, '18.00', '14.00', '77.00', '100.00', '21:28:53', '2022-08-22'),
(9, '18.00', '14.00', '77.00', '100.00', '21:29:53', '2022-08-22'),
(10, '18.00', '14.00', '78.00', '100.00', '21:30:54', '2022-08-22');

--

-- Índices para tablas volcadas
--

--

-- Indices de la tabla `lectura_ambiente`
--
ALTER TABLE `lectura_ambiente`
  ADD PRIMARY KEY (`nRegistro`);

--

-- AUTO_INCREMENT de las tablas volcadas
--

--

-- AUTO_INCREMENT de la tabla `lectura_ambiente`
--
ALTER TABLE `lectura_ambiente`
  MODIFY `nRegistro` int(10) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=561;
COMMIT;

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;

```