Garden Smart (DAD Project)

INTRODUCCIÓN

En el contexto actual de la agricultura y la jardinería, la automatización y la monitorización juegan un papel fundamental e la optimización del cuidado de las plantas. Con el avance de la tecnología de sensores y actuadores, se ha vuelto posible diseñar sistemas inteligentes que puedan monitorizar las condiciones ambientales y responder de manera autómata para mantener un entorno óptimo para el crecimiento de las plantas. En este proyecto, se ha desarrollado una solución de automatización para el riesgo de plantas utilizando un ESP32, un sensor de temperatura/humedad y un servo como actuador.

El objetivo principal de este trabajo era implementar un sistema para la monitorización del jardín. En cambio, hasta la fecha, solo hemos podido implementar el dispositivo para el riego automático de plantas. Esto lo hemos realizado mediante la integración de un sensor de temperatura/humedad que proporciona mediciones precisas del entorno, y un servo que actúa como un mecanismo de riego controlado electrónicamente. A través de la programación de la placa, se ha desarrollado una funcionalidad que analiza los datos del sensor y activa el servo según las condiciones predefinidas.

Este proyecto no solo presenta una solución práctica para el riego automatizado de plantas, sino que también demuestra la aplicación de Internet de las cosas (IoT) en el ámbito de la jardinería.

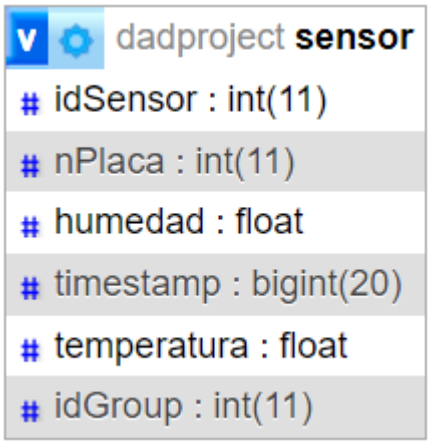
En esta memoria, se detallará el diseño e implementación del sistema de riego inteligente, incluyendo la selección de componentes, el diseño del circuito, el desarrollo del software, y las pruebas realizadas para validar el funcionamiento del sistema.

FUNCIONALIDAD

- Automatizar el riego: esta funcionalidad implica la capacidad del sistema para controlar automáticamente el riego de las plantas en respuesta a las lecturas del sensor de humedad/temperatura. Cuando las condiciones indican que las plantas necesitan agua (temperatura>28º), el sistema activa un servo motor que actúa como mecanismo de riego (abre el flujo de agua). Si la temperatura fuera menor de 28º, lo cerraría.
  El proceso de automatización se basa en la programación del sistema para que compare continuamente las lecturas del sensor con ciertos umbrales predefinidos.
- Ofrecer registro temporal de los datos recogidos: Esta función implica la capacidad del sistema para almacenar los datos de temperatura y humedad recopilados en diferentes momentos a lo largo del tiempo. Para ello, se utiliza una base de datos en función de la marca de tiempo que se recogieron. Por ejemplo, cada vez que se toma una lectura del sensor de temperatura/humedad, se registra junto con su timestamp correspondiente.
  Este registro temporal de los datos permite realizar un seguimiento histórico de las condiciones ambientales del jardín. Al tener acceso a registros pasados, los usuarios pueden analizar patrones climáticos y detectar cambios estacionales.

BASE DE DATOS

La base de datos diseñada para nuestro proyecto es relativamente simple, puesto que la única información que almenaremos será la de los sensores y actuadores. Para ello, hemos creado una tabla para cada uno de ellos. Los sensores solo precisan del id del sensor, el número de Placa, el timestamp, el valor de humedad y el valor de la temperatura. Por otro lado, los actuadores tienen como atributos el número de Placa, el id del actuador, el timestamp y valores booleanos para activo y encendido.



Para las tablas hemos usado atributos básicos con el objetivo de obtener una base de datos simple y que no nos lleve una gran cantidad de tiempo trabajar con ella.

API REST

En el diseño de la API REST principalmente hemos usado funciones básicas que nos conecten la base de datos. Nuestra API REST está formada por:

- **Métodos GET**

  Hemos creado varios métodos GET, que podemos usar para obtener los valores de los sensores y actuadores guardados en la base de datos.

- **Métodos POST**

  Para insertar nuevos elementos hemos usado los métodos POST. Hemos implementado dos funciones POST, “addSensor” y “addActuador”, de forma que estos datos se añaden en las tablas correspondientes de la base de datos.

- **Métodos PUT**

  Hemos usado métodos PUT para actualizar los registros. Para ello hemos creado las funciones “updateSensor” y “updateActuador”.

- **Métodos DELETE**

  Hemos usado métodos DELETE para borrar registros de la base de datos. Para ello hemos creado las funciones “deleteSensor” y “deleteActuador”.

Para estas peticiones hemos usado URLs intuitivas y cortas. La nomenclatura es "/api/(tipo de elemento)/: (id)" independientemente del tipo de petición, puesto que esto no supone un problema, pero ara algunas peticiones es distinta:

- Para obtener datos relacionados con los sensores, en '(tipo de elemento)' colocamos "sensor". Si por el contrario, queremos tratar con los actuadores colocaremos "actuador"
- Cuando queremos obtener todos los sensores o todos lo actuadores, la nomenclatura será:
  "/api/(tipo de elemento)/all".
- Si solamente queremos obtener los últimos sensores, la nomenclatura será:
  "/api/sensor/last" o "api/sensor/group/last"

Por último, en cuanto al cuerpo de las peticiones para los métodos POST, simplemente usamos un JSON para las columnas en la base de datos, aunque es necesario aclarar que el cuerpo debe contener todos los atributos, de lo contrario no será efectiva la petición. Esto provoca que debamos introducir también el id en el cuerpo. El id será ignorado, puesto que, en los POST, la BBDD es quien lo asigna.

MQTT

El uso de MQTT en nuestro proyecto es corto pero importante, ya que su uso es necesario para un sistema de actuadores eficiente. En nuestro caso, solo nos será necesario el uso de un canal para nuestro actuador (un topic por placa).

Hemos usado esta tecnología conectando el servidor (en nuestra API Rest) a un bróker MQTT y suscribiendo la placa a un topic concreto, que será el identificador de la placa a la hora de escalar el proyecto a utilizar varias de ellas.

Usando esta suscripción, cuando la placa detecta un valor de temperatura superior a un umbral, manda una señal de "ON" al topic. Si es inferior a ese umbral, se enviará una señal de "OFF". A raíz de esto, hemos implementado la funcionalidad pertinente para que el actuador (en nuestro caso el servo-motor) cambie de posición en función de lo que se reciba en el topic al que está suscrita la placa.

ESP32

El código para el ESP32 sigue una estructura en la que la mayoría de los casos solo debemos adaptarla a nuestro proyecto:

- **Setup**: iniciamos la conexión Wifi e inicializamos el cliente MQTT, y después obtenemos los IDs de los sensores y actuadores del dispositivo para las peticiones. Además, inicializamos el servo-motor.
- **Loop**: en caso de no estar conectados al servidor MQTT nos volvemos a conectar y luego mediante la función loop() se controla un contador para realizar lecturas de sensores ya acciones con un servo a intervalos definidos. También se maneja la conexión MQTT en cada interacción del bucle llamando a la función HandleMqtt(). Los datos obtenidos en cada interacción se envían a un servidor.

- **Funciones MQTT**: Tenemos la función Callback, que recibe el mensaje de MQTT y comprueba si es diferente al que había antes. A partir de esto, maneja el comportamiento del servo-motor.
- **Manejo de HTTP:** El código incluye varias funciones para manejar solicitudes HTTP, tanto para enviar datos (POST) como para recibir datos (GET). Estas funciones incluyen la serialización y deserialización de datos en formato JSON para la comunicación con el servidor.

En el bucle principal, cada vez que el contador alcanza 1000, se realiza una lectura del sensor y se envían los datos al servidor. El bucle también maneja la conexión MQTT en cada iteración.