



SISTEMA DE RIEGO AUTOMÁTICO

Desarrollo de Aplicaciones Distribuidas



ÁLVARO GARCÍA RODRÍGUEZ
MARCOS STEPHAN PERALVO GERMAN

Contenido

1.- PRIMERA ITERACIÓN	2
1.1.- Introducción.....	2
2.- SEGUNDA ITERACIÓN (BBDD).....	3
2.1.- UML	3
2.2.- ESQUEMA E/R	4
3.- TERCERA ITERACIÓN (API Rest)	4
3.1.- POST	4
3.2.- GET	9
3.3.- PUT	13
3.4.- DELETE	15
4.- CUARTA ITERACIÓN (MQTT).....	17
4.1.- CANAL SENSOR.....	17
4.2.- CANAL INFO	17
4.3.- CANAL RIEGO	17
5.- QUINTA ITERACIÓN (ESP8266)	18
5.1.- Integración con la API Rest	18
Funciones setup y loop:.....	18
Función que leerá el sensor:.....	18
Función que obtendrá el umbral de humedad para activar la bomba de agua:	20
Función que hará la acción de regar:	20
5.2.- Integración con MQTT	21
Función setup y loop:	21
Función callback:.....	22
Función cliente MQTT:	23
Función reconnect:.....	23
Función topicoSensor:	24
Función topicoInfo:	24

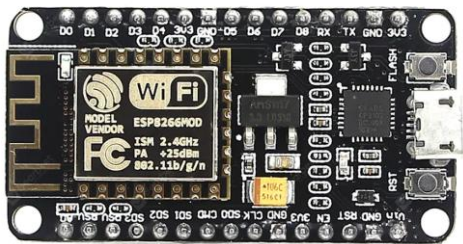
1.- PRIMERA ITERACIÓN

1.1.- Introducción

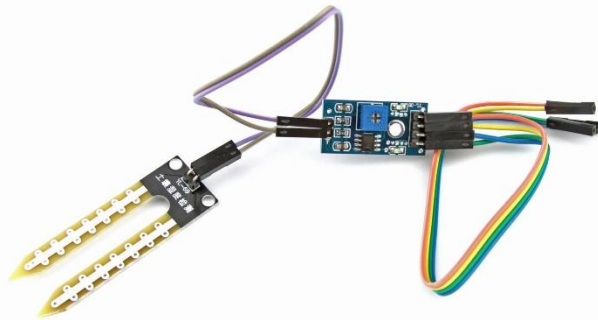
El proyecto por realizar consistirá en un sistema de riego automático.

Para la realización de este, se estima que se utilizará una placa WiFi NodeMCU ESP8266, un sensor de humedad, una mini bomba de agua, un relé para controlar la potencia de la bomba, unas resistencias y tres diodos LEDs para informar al usuario del estado de la tierra a regar: amarillo si la tierra está lista para ser regada, verde si la tierra está húmeda, rojo y parpadeando, no está el sensor en la tierra.

A lo largo de la realización del proyecto, se verá si implementar alguna variación de este y/o algunas recomendaciones del profesor.



NodeMCU ESP8266



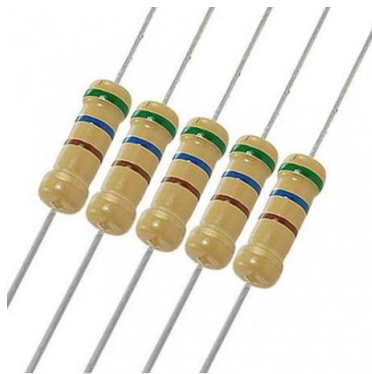
Sensor de humedad Arduino



Mini bomba de agua



Diodos LEDs



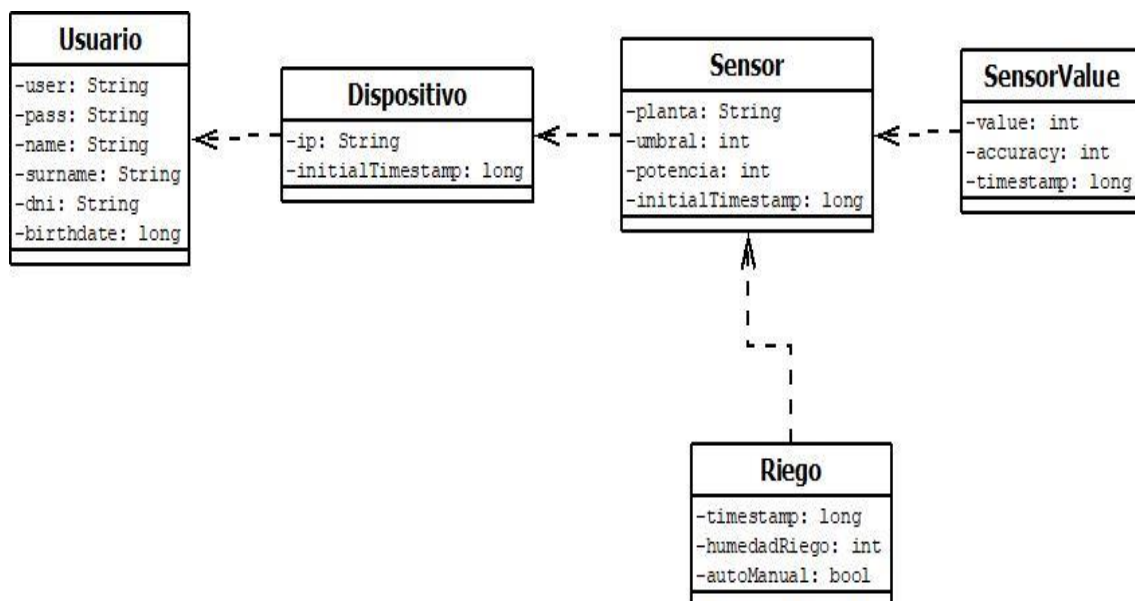
Resistencias



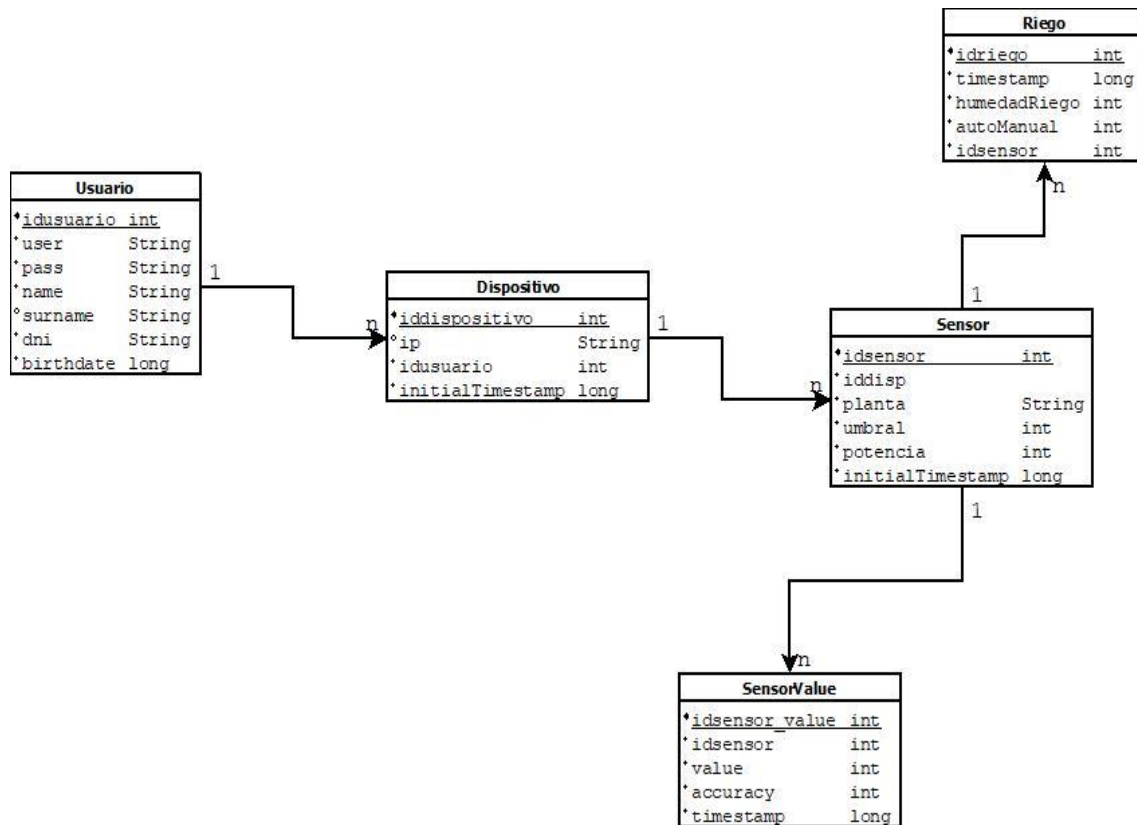
Relé 3,3 V

2.- SEGUNDA ITERACIÓN (BBDD)

2.1.- UML



2.2.- ESQUEMA E/R



3.- TERCERA ITERACIÓN (API Rest)

3.1.- POST

Usaremos los POST para las inserciones en la base de datos.

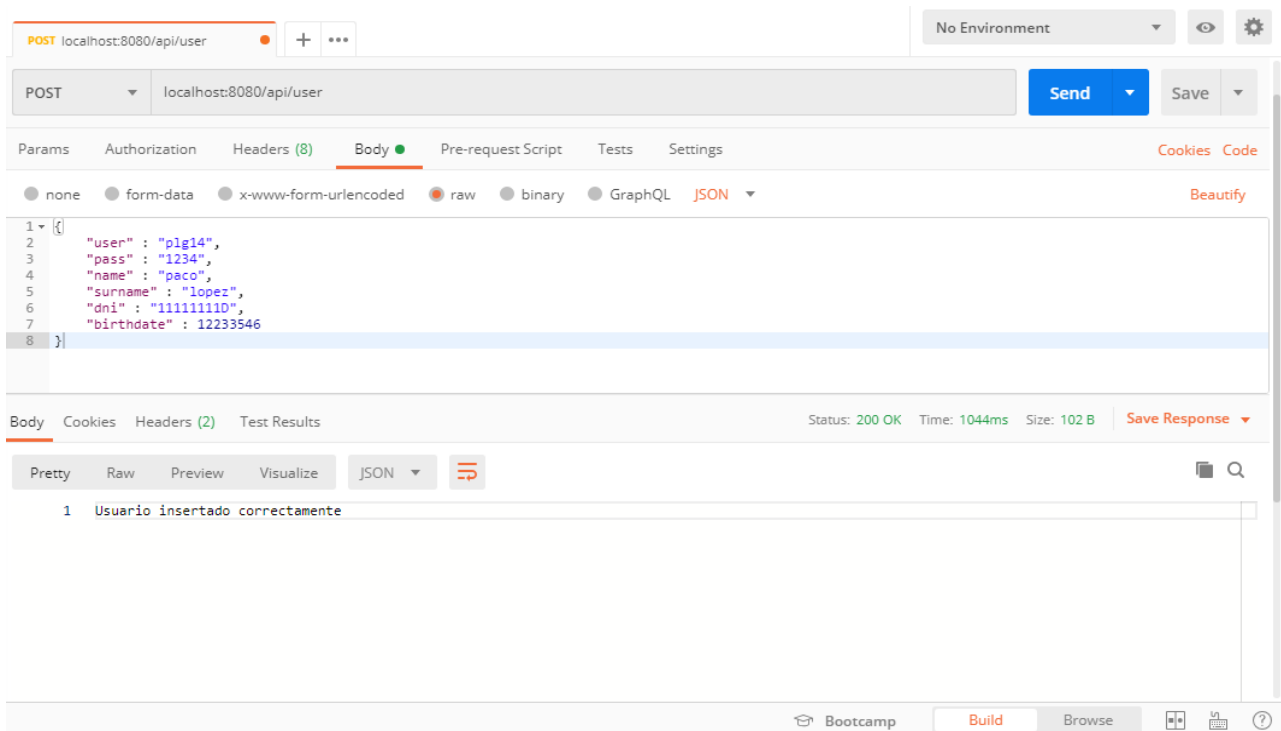
■ "/api/user"

Insercción en la base de datos de un nuevo usuario (registro). No se le pasa ningún parámetro.

El cuerpo es el siguiente (ejemplo):

```
{
  "user" : "plg14",
  "pass" : "1234",
  "name" : "paco",
  "surname" : "lopez",
  "dni" : "11111111D",
  "birthdate" : 12233546
}
```

En cuanto se inserte en la base de datos, una respuesta al cliente sería que el registro se ha completado satisfactoriamente.



■ "/api/device"

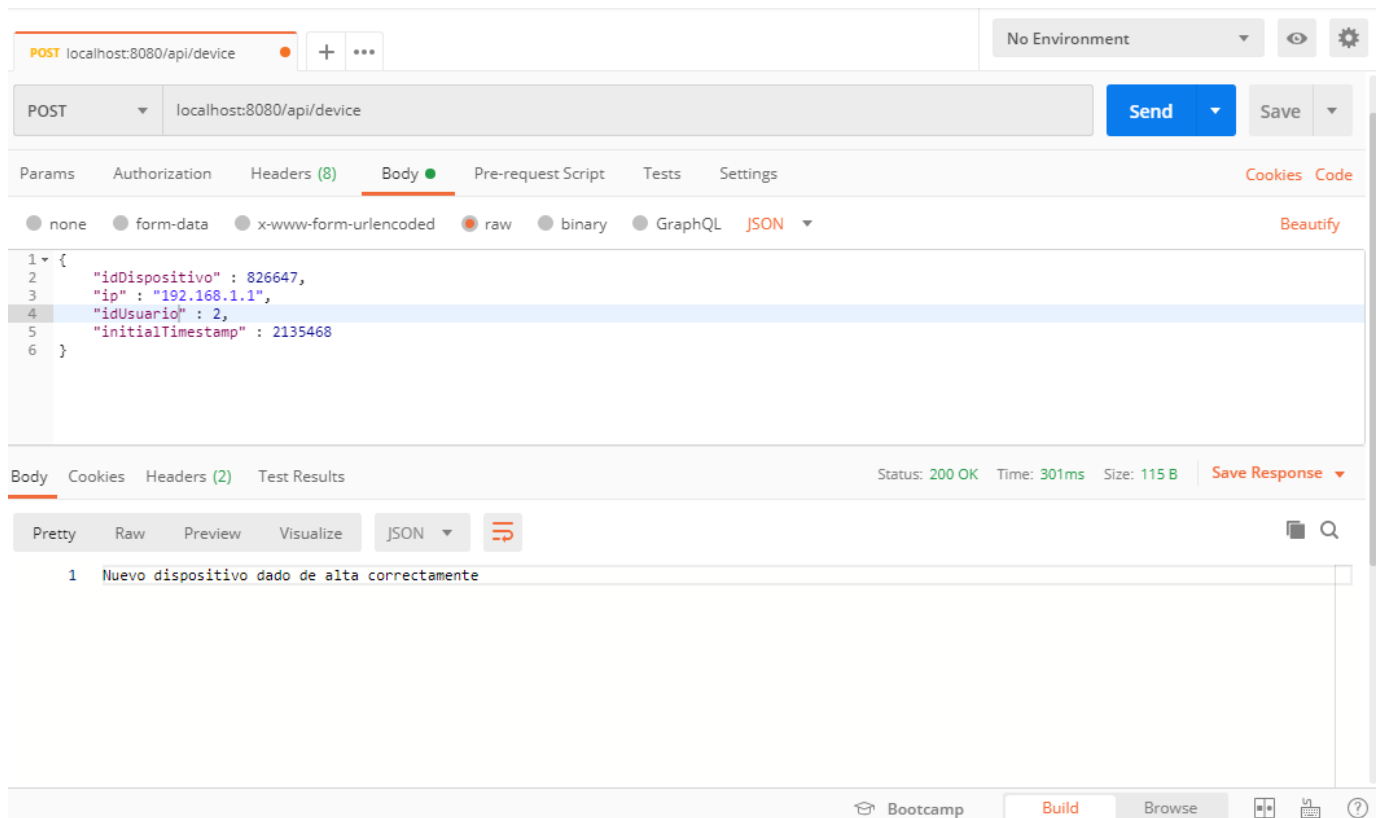
Se llamará a esta función cuando un usuario quiere dar de alta a un nuevo dispositivo (ESP8266).

El cuerpo es el siguiente:

```
{
  "idDispositivo" : 826647,
  "ip" : "192.168.1.1",
  "idUsuario" : 2,
  "initialTimestamp" : 2135468
}
```

El id se pondrá de forma manual para identificar siempre el dispositivo físico.

El mensaje al usuario sería que el dispositivo se ha dado de alta correctamente.



■ `"/api/device/sensor"`

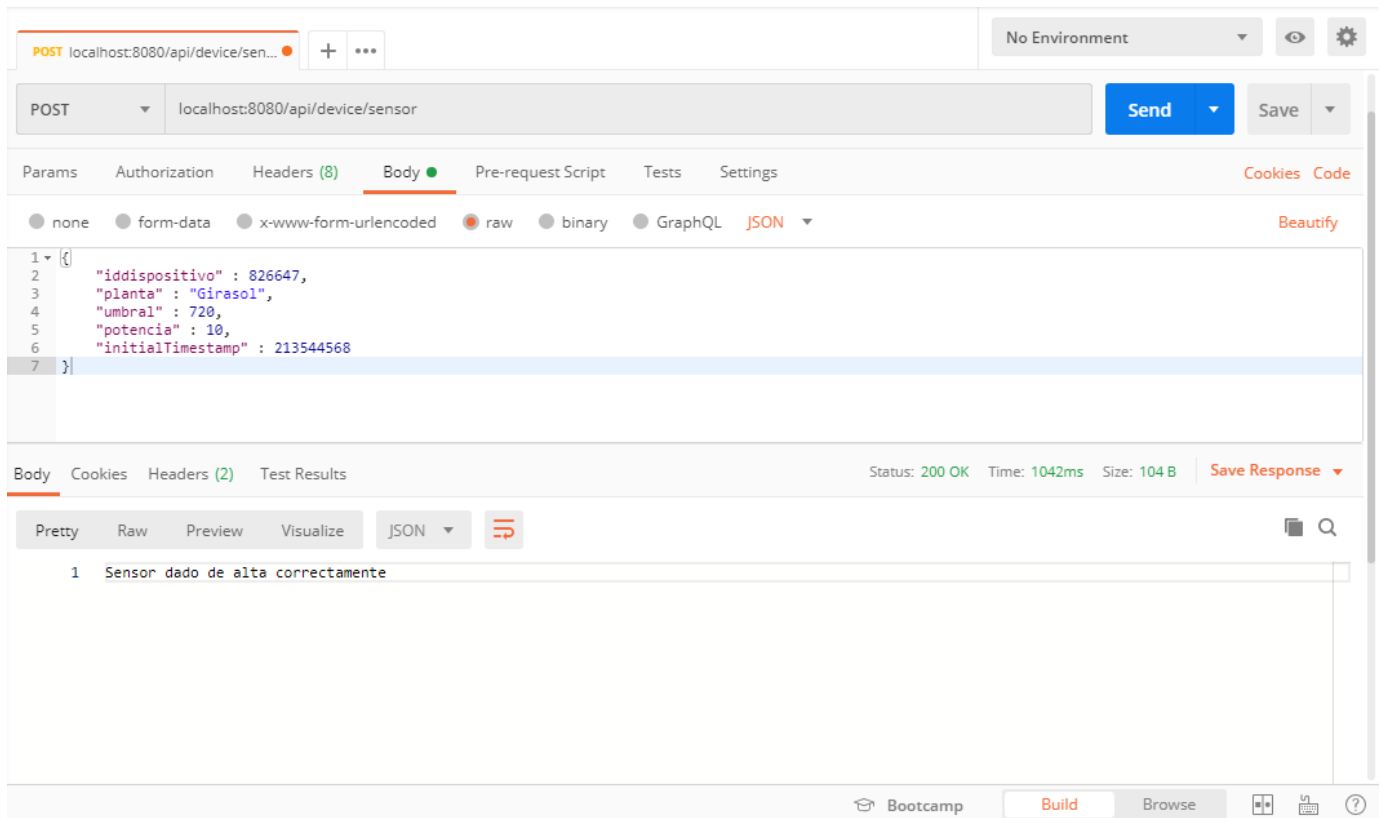
Cuando el usuario conecte un sensor al dispositivo, dicho usuario dará de alta al sensor.

El cuerpo es el siguiente:

```
{  "iddispositivo": 826647,  "planta": "Girasol",  "umbral": 720,  "potencia": 10,  "initialTimestamp": 213544568}
```

Se tiene que especificar a qué planta estará conectado el sensor, umbral que de llegar el sensor activará la bomba de agua, y la potencia por la que funcionará la bomba.

La respuesta al cliente será que el sensor se ha dado de alta correctamente.



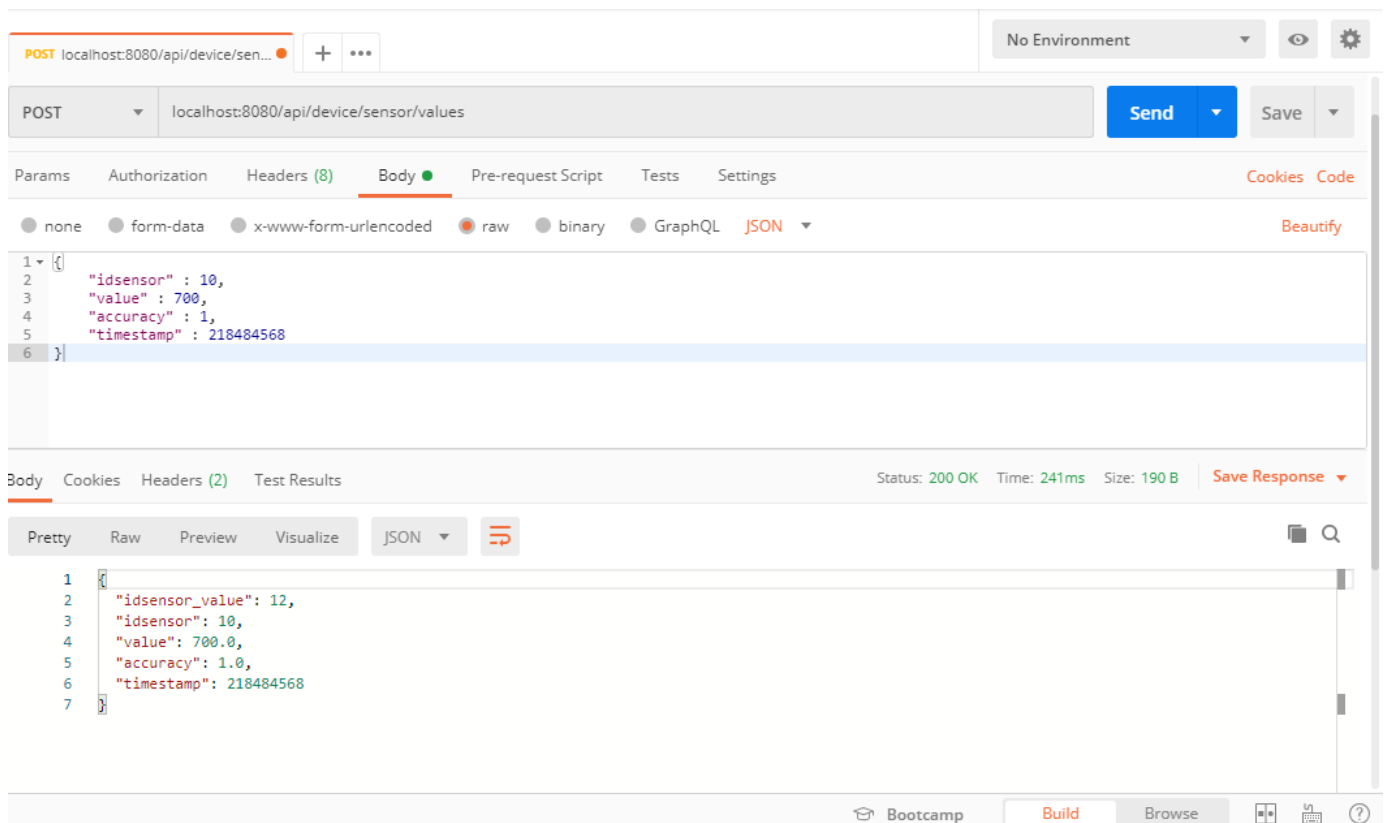
■ "/api/device/sensor/values"

Este método se llamará cuando el sensor haga lecturas de la humedad.

El cuerpo es el siguiente:

```
{
  "idsensor" : 10,
  "value" : 700,
  "accuracy" : 1,
  "timestamp" : 218484568
}
```

No habrá respuesta al usuario.



■ `/api/device/sensor/riego`

Este método se llamará cada vez que se realice la acción de riego, para guardar un historial de los riegos.

El cuerpo es el siguiente:

```
{
  "timestamp" : 654654321,
  "humedad" : 650,
  "manualAuto" : 0,
  "idsensor" : 10
}
```

El campo `humedad` es la humedad registrada cuando se regó y el campo `manualAuto` es 0 si es un riego manual o 1 si ha sido automático.

No habrá respuesta al usuario.

The screenshot shows a REST client interface with a POST request to `localhost:8080/api/device/sensor/riego`. The request body is a JSON object:

```
{  "timestamp": 654654321,  "humedad": 650,  "manualAuto": 0,  "idsensor": 10}
```

The response is a 200 OK status with a JSON body:

```
{  "id": 0,  "timestamp": 654654321,  "humedad": 650,  "manualAuto": false,  "idsensor": 10}
```

3.2.- GET

Usaremos los GET para hacer SELECT a la BBDD.

- `"/api/user/:user/:pass"`

Esta URL se usará para el login del usuario. Se pedirá el usuario y la contraseña, rellena en el formulario de login. Si es correcto, se accederá a la aplicación, si es incorrecto, se enviará un mensaje de error.

Como es un GET, no hay cuerpo.

GET localhost:8080/api/user/plg14/...

Untitled Request

GET localhost:8080/api/user/plg14/1234

Send Save

Params Authorization Headers (8) Body ● Pre-request Script Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (2) Test Results Status: 200 OK Time: 1076 ms Size: 228 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "id": 5,
4     "user": "plg14",
5     "pass": "1234",
6     "name": "paco",
7     "surname": "lopez",
8     "dni": "11111111D",
9     "birthdate": 12233546
10  }
11 }
```

■ "/api/user/:idusuario"

Dado el id de un usuario, carga todos los datos de dicho usuario. Usado para cargar las cajas de texto de edición de usuario. Como es un GET, no hay cuerpo.

GET localhost:8080/api/user/5

Untitled Request

GET localhost:8080/api/user/5

Send Save

Params Authorization Headers (8) Body ● Pre-request Script Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (2) Test Results Status: 200 OK Time: 20 ms Size: 228 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "id": 5,
4     "user": "plg14",
5     "pass": "1234",
6     "name": "paco",
7     "surname": "lopez",
8     "dni": "11111111D",
9     "birthdate": 12233546
10  }
11 }
```

■ "/api/device/:idusuario"

Dado el id del usuario, permite visualizar una lista de dispositivos que tiene dicho usuario dado de alta. Como es un GET, no tiene cuerpo.

GET localhost:8080/api/device/2

Send Save

Params Authorization Headers (8) Body ● Pre-request Script Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (2) Test Results Status: 200 OK Time: 30 ms Size: 300 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "idDispositivo": 82665,
4     "ip": "192.168.1.1",
5     "idUsuario": 2,
6     "initialTimestamp": 5655654654
7   },
8   {
9     "idDispositivo": 826647,
10    "ip": "192.168.1.1",
11    "idUsuario": 2,
12    "initialTimestamp": 2135468
13  }
14 }
```

Bootcamp Build Browse

■ "/api/device/sensor/:idsensor"

Dado el id del dispositivo, carga una lista de sensores vinculados a dicho dispositivo. Como es un GET, no tiene cuerpo.

GET localhost:8080/api/device/sens... Send Save

Params Authorization Headers (8) Body ● Pre-request Script Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (2) Test Results Status: 200 OK Time: 976 ms Size: 221 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "id": 10,
4     "iddispositivo": 826647,
5     "planta": "Girasol",
6     "umbral": 720,
7     "potencia": 10,
8     "initialTimestamp": 213544568
9   }
10 }
```

Bootcamp Build Browse

■ "/api/device/sensor/values/:idSensor"

Dado el id del sensor, mostrará una tabla con todos los valores leídos por dicho sensor. Como es un GET, no tiene cuerpo.

The screenshot shows a REST client interface with the following details:

- Request:** GET `localhost:8080/api/device/sensor/values/10`
- Response Status:** 200 OK, Time: 30 ms, Size: 194 B
- Response Body (JSON):**

```
{  "idsensor_value": 12,  "idsensor": 10,  "value": 700.0,  "accuracy": 1.0,  "timestamp": 218484568}
```

■ "/api/device/sensor/riego/:idsensor"

Dado el id del sensor, mostrará un registro de los riegos que se han realizado dicho sensor. Como es un GET, no lleva cuerpo.

The screenshot shows a REST client interface with the following details:

- Request:** GET `localhost:8080/api/device/sensor/riego/10`
- Response Status:** 200 OK, Time: 22 ms, Size: 185 B
- Response Body (JSON):**

```
{  "id": 4,  "timestamp": 654654321,  "humedad": 650,  "manualAuto": false,  "idsensor": 10}
```

3.3.- PUT

Usaremos los PUT para hacer UPDATE en la BBDD.

■ "/api/user/:idusuario"

Dado el id del usuario, se actualizarán los datos de dicho usuario con los datos rellenados en un formulario.

El cuerpo es el siguiente:

```
{
  "user" : "plg225",
  "pass" : "2563",
  "name" : "Paco",
  "surname" : "Lopez",
  "dni" : "11111111D",
  "birthdate" : 21355546
}
```

Como se puede ver, se permite cambiar cualquier dato del usuario.

Una respuesta al cliente sería que los datos del usuario se han actualizado correctamente.

The screenshot shows a REST client interface with the following details:

- Request:** PUT localhost:8080/api/user/5
- Body:** A JSON object with the following fields: "user": "plg225", "pass": "2563", "name": "Paco", "surname": "Lopez", "dni": "11111111D", "birthdate": 21355546.
- Response:** Status: 200 OK, Time: 927 ms, Size: 103 B. The response body is "1 Datos actualizados correctamente".

■ "/api/device/sensor/:idsensor"

Dado el id del sensor, se actualizarán los datos del sensor, por si cambias de planta por ejemplo.

El cuerpo es el siguiente:

```
{  
  "planta" : "Aloe",  
  "umbral" : 650,  
  "potencia" : 5  
}
```

Una respuesta al cliente sería que los datos han sido actualizados correctamente.

The screenshot shows a REST client interface with the following details:

- Request:** PUT localhost:8080/api/device/sensor/10
- Body:**

```
{  
  "planta" : "Aloe",  
  "umbral" : 650,  
  "potencia" : 5  
}
```
- Response:** Status: 200 OK, Time: 1020 ms, Size: 114 B. The response body is:

```
1 Datos del sensor actualizados correctamente
```

3.4.- DELETE

Usaremos los métodos DELETE para hacer DELETE en la BBDD.

■ `"/api/device/sensor/:idsensor"`

Dado el id del sensor, se borrará todo dato de dicho sensor. Como está configurado para un borrado en CASCADE, las tablas de los valores del sensor y de los riegos se borrarán.

Como sólo nos interesa el id, no habrá cuerpo. Un mensaje al cliente sería que el sensor se ha dado de baja correctamente.

The screenshot shows a REST client interface with the following details:

- Request:** Method `DELETE`, URL `localhost:8080/api/device/sensor/10`.
- Response:** Status `200 OK`, Time `864 ms`, Size `104 B`. The response body is `1 Sensor dado de baja correctamente`.
- Interface Elements:** Tabs for Params, Authorization, Headers (8), Body, Pre-request Script, Tests, Settings. A table for Query Params with columns KEY, VALUE, and DESCRIPTION. A bottom bar with buttons for Pretty, Raw, Preview, Visualize, JSON, and a search icon.

■ `"/api/device/:iddispositivo"`

Dado el id del dispositivo, se borrará todo dato de dicho dispositivo. Como está configurado para un borrado en CASCADE, los datos de todos los sensores conectados al dispositivo, se borrarán.

Como sólo nos interesa el id, no habrá cuerpo. Un mensaje al cliente sería que el dispositivo se ha dado de baja correctamente.

DEL localhost:8080/api/device/8266...

No Environment

Untitled Request

Comments 0

DELETElocalhost:8080/api/device/826647

Send

Save

ParamsAuthorizationHeaders (8)Body●Pre-request ScriptTestsSettingsCookiesCode

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

BodyCookiesHeaders (2)Test Results

Status: 200 OKTime: 620 msSize: 109 BSave Response

PrettyRawPreviewVisualizeJSON

1 Dispositivo dado de baja correctamente

BootcampBuildBrowse

■ "/api/user/:idusuario"

Dado el id del usuario, se borrará todo dato de dicho usuario. Como está configurado para un borrado en CASCADE, los datos de todos los dispositivos asociados al usuario, se borrarán.

Como sólo nos interesa el id, no habrá cuerpo. Un mensaje al cliente sería que se ha dado de baja correctamente.

DEL localhost:8080/api/user/5

No Environment

Untitled Request

Comments 0

DELETElocalhost:8080/api/user/5

Send

Save

ParamsAuthorizationHeaders (8)Body●Pre-request ScriptTestsSettingsCookiesCode

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

BodyCookiesHeaders (2)Test Results

Status: 200 OKTime: 733 msSize: 103 BSave Response

PrettyRawPreviewVisualizeJSON

1 Se ha dado de baja correctamente

BootcampBuildBrowse

4.- CUARTA ITERACIÓN (MQTT)

4.1.- CANAL SENSOR

El propósito de este canal es recibir todas las lecturas de los sensores que el módulo ESP8266 recoge, así si un cliente se suscribe a este canal podrá tener acceso a las lecturas del sensor.

El formato de mensaje será un JSON que constará con los siguientes parámetros:

- Idsensor: el id del sensor que haya hecho la lectura.
- Value: valor de la lectura del sensor.
- Timestamp: indica la hora en la que se realizó la lectura.
- Accuracy: representa el porcentaje de error que el sensor tiene al realizar una lectura.

Ejemplo:

```
{
  "idsensor" : 10,
  "value" : 650,
  "timestamp" : 1587252138,
  "accuracy" : 1
}
```

4.2.- CANAL INFO

En este canal el módulo ESP8266 se encarga de enviar mensajes en intervalos de tiempo sobre el estado actual de los sensores para saber si está en funcionamiento o no.

El formato de mensaje será de texto plano que constará con un mensaje que nos diga el id del sensor y su estado actual:

“El sensor con id 1 está capturando información”

“Error!!! el sensor con id 5 no responde”

4.3.- CANAL RIEGO

El propósito de este canal es que un cliente envíe un mensaje el cual será recibido por el modulo ESP8266 el cual hará un riego de agua al sensor que se haya indicado en el mensaje si la planta en cuestión no tiene una humedad demasiado alta para evitar dañar a la planta, esto se indicara con un mensaje de respuesta que indicara si el riego se realizó o no.

El formato del mensaje que va a ser recogido por el módulo será un JSON en donde se indique el id del sensor que va a realizar el riego.

Cuando el modulo reciba este mensaje se encarga de realizar el riego solo si se detecta que la humedad actual no es demasiado alta en comparación al umbral de humedad que este configurado, en este caso se responde con un mensaje de texto plano que diga que la operación se ha realizado:

“El riego se ha realizado exitosamente”

En caso contrario se debe enviar una respuesta que diga que no ha podido realizar el riego:

“Peligro! El riego ha fallado, la humedad actual de la planta es muy alta y se ha cancelado el riego para evitar daños a la planta”

5.- QUINTA ITERACIÓN (ESP8266)

5.1.- Integración con la API Rest

Funciones setup y loop:

```
void setup() {  
  Serial.begin(9600);  
  WiFi.begin(SSID, PASS);  
  Serial.print("Connecting...");  
  while(WiFi.status() != WL_CONNECTED){  
    delay(500);  
    Serial.print(".");  
  }  
  Serial.print("Connected, IP address: ");  
  Serial.print(WiFi.localIP());  
}  
  
void loop() {  
  lecturaSensor(1); //pasarle el numero de pin que esta conectado el sensor al ESP  
  delay(10000);  
}
```

Función que leerá el sensor:

```
void lecturaSensor(int pin){  
  if(WiFi.status() == WL_CONNECTED){  
    HTTPClient http;  
    http.begin(client, SERVER_IP, SERVER_PORT, "/api/device/sensor/values", true);  
    http.addHeader("Content-Type", "application/json");  
  
    const size_t capacity = JSON_OBJECT_SIZE(3) + JSON_ARRAY_SIZE(2) + 60;  
    DynamicJsonDocument doc(capacity);
```

```

doc["idsensor"] = pin;
doc["value"] = valor; //aqui ira en vez de valor, la lectura del sensor
doc["accuracy"] = 1;
doc["timestamp"] = 12052020130800; //funcion para calcular timestamp actual

int umbral = getUmbral(pin);
delay(1000);

Serial.println(umbral);

String output;

serializeJson(doc, output);

http.POST(output);

String payload = http.getString();

Serial.println("Resultado: lectura realizada");

if(valor >= umbral){
    regar(pin, false); //false porque no se ha accionado el boton de riego;
    valor = 495;
}
valor++;
}
}

```

Función que obtendrá el umbral de humedad para activar la bomba de agua:

```
int getUmbral(int id){
    int umbral = -1;
    if(WiFi.status() == WL_CONNECTED){
        HTTPClient http;
        http.begin(client, SERVER_IP, SERVER_PORT, "/api/device/sensor/id/"+String(id), true);
        http.GET();

        String payload = http.getString();

        const size_t capacity = JSON_OBJECT_SIZE(6) + JSON_ARRAY_SIZE(2) + 60;
        DynamicJsonDocument doc(capacity);

        DeserializationError error = deserializeJson(doc,payload);

        if(error){
            Serial.print("deserializeJson() failed: ");
            Serial.println(error.c_str());
        }
        umbral = doc[0]["umbral"].as<int>();
    }

    return umbral;
}
```

Función que hará la acción de regar:

```
void regar(int pin, boolean pulsador){
    if(WiFi.status() == WL_CONNECTED){
        HTTPClient http;
        http.begin(client, SERVER_IP, SERVER_PORT, "/api/device/sensor/riego", true);
```

```

http.addHeader("Content-Type", "application/json");

const size_t capacity = JSON_OBJECT_SIZE(3) + JSON_ARRAY_SIZE(2) + 60;
DynamicJsonDocument doc(capacity);
doc["timestamp"] = 12052020130800;
doc["humedad"] = valor; //valor leido por el sensor
doc["manualAuto"] = pulsador; //estado del boton
doc["idsensor"] = pin;

String output;
serializeJson(doc, output);

http.POST(output);

String payload = http.getString();

Serial.println("Resultado: " + payload);
}
}

```

5.2.- Integración con MQTT

Función setup y loop:

```

void setup() {
  Serial.begin(9600);

  WiFi.begin(SSID, PASS);

  Serial.print("Connecting...");

  while(WiFi.status() != WL_CONNECTED){
    delay(500);

```

```
Serial.print(".");
}
```

```
Serial.print("Connected, IP address: ");
```

```
Serial.print(WiFi.localIP());
```

```
client.setServer(mqtt_server, 1885); //indica el servido MQTT y el puerto
```

```
client.setCallback(callback); //Indica funcion que se encarga de recibir mensajes desde los canales
```

```
}
```

```
void loop() {
```

```
    clienteMqtt();
```

```
    delay(5000);
```

```
}
```

Función callback:

```
void callback(char* topic, byte* payload, unsigned int length) {
```

```
    if (strcmp(topic, "riego")==0) { //Comprueba si se recibe un mensaje desde el topico riego
```

```
        Serial.println("Mensaje recibido desde topico Riego");
```

```
        StaticJsonDocument<256> doc;
```

```
        deserializeJson(doc, payload, length);
```

```
        char buffer[512];
```

```
        if(valor>=getUmbral(doc["idsensor"].as<int>())) //Si el valordel sensor es mayor al umbral se hace un riego
```

```
{
```

```
    regar(doc["idsensor"].as<int>(), true);
```

```
    sprintf(buffer, "El riego se ha realizado exitosamente");
```

```
}else { //Si no lo es se envia un mensaje de error
```

```
    sprintf(buffer, "Peligro! El riego ha fallado, valor < umbral");
```

```
}
```

```

    client.publish("riego", buffer);//publica mensaje en el canal riego
}
}

```

Función cliente MQTT:

```

void clienteMqtt(){
    if (!client.connected()) { //Comprueba si estamos conectados al servidor clienteMqtt
        do {
            Serial.print("Connecting ...\n");
        } while(reconnect()==0);
    }
    else {

        topicoSensor(1);//Funcion que publica mensajes en el topico sensor
        delay(1000);

        topicoInfo(1);//Funcion que publica mensajes en el topico info
        delay(1000);
    }
}

```

Función reconnect:

```

boolean reconnect() { //Funcion que se subscribe a los canales
    if (client.connect("Arduino client")) {
        client.subscribe("info");
        Serial.println("conectado a topico Info");
        client.subscribe("sensor");
        Serial.println("conectado a topico Sensor");
        client.subscribe("riego");
        Serial.println("conectado a topico Riego");
    }
}

```



```

        return client.connected();
    }
    Serial.println("Subscripcion fallida");
    return 0;
}

```

Función `topicoSensor`:

```

void topicoSensor(int pin){//funcion que construye el json que se va a publicar en el canal

    const size_t capacity = JSON_OBJECT_SIZE(3) + JSON_ARRAY_SIZE(2) + 60;
    DynamicJsonDocument doc(capacity);

    doc["idsensor"] = pin;
    doc["value"] = valor; //aqui ira en vez de valor, la lectura del sensor
    doc["accuracy"] = 1;
    doc["timestamp"] = 12052020130800; //funcion para calcular timestamp actual

    delay(1000);

    char buffer[512];
    serializeJson(doc, buffer);
    client.publish("sensor", buffer);//publica el json en el canal
    Serial.println("Se ha publicado en el topico Sensor");
}

```

Función `topicoInfo`:

```

void topicoInfo(int pin){//funcion que publica el estado de los sensores

    char buffer[512];
    if(valor!=-1){
        sprintf(buffer,"El sensor con id %d está capturando información",pin);
    }else{

```

```
    sprintf(buffer,"Error!!! el sensor con id %d no responde",pin);  
}  
  
client.publish("info", buffer);//publica el mensaje en el canal  
Serial.println("Se ha publicado en el topico Info");  
}
```