



SISTEMA DE RIEGO AUTOMÁTICO

Desarrollo de Aplicaciones Distribuidas



ÁLVARO GARCÍA RODRÍGUEZ
MARCOS STEPHAN PERALVO GERMAN

Contenido

1.- PRIMERA ITERACIÓN	3
1.1.- Introducción.....	3
2.- SEGUNDA ITERACIÓN (BBDD)	4
2.1.- UML.....	4
2.2.- ESQUEMA E/R	4
3.- TERCERA ITERACIÓN (API Rest).....	5
3.1.- POST	5
3.2.- GET	9
3.3.- PUT.....	13
3.4.- DELETE.....	14
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:.....	19
Función que hará la acción de regar:	19
5.2.- Integración con MQTT	20
Función setup y loop:	20
Función callback:	20
Función cliente MQTT:	21
Función reconnect:.....	21
Función topicoSensor:.....	21
Función topicoInfo:	22
6.- ENTREGA FINAL.....	23
6.1.- Variables Globales:.....	26
6.2.- Funciones Auxiliares:	27
6.2.1.- Función guardarLecturas():	27
6.2.2.- Función regar():.....	28
6.2.3.- Función getUmbral():	29
6.2.4.- Función estadoLEDs():.....	29
6.2.5.- Función getPotencia():	30

6.2.6.- Función riegoManual():.....	30
6.3.- Función Void Setup():.....	31
6.4.- Función Void Loop():	31

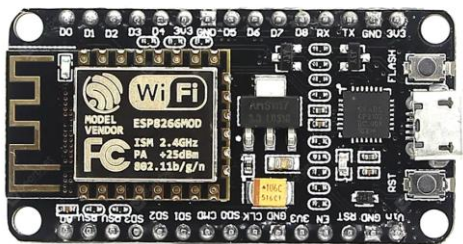
1.- PRIMERA ITERACIÓN

1.1.- Introducción

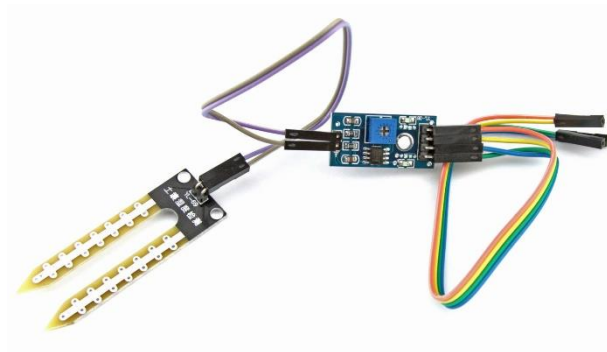
El proyecto por realizar consistirá en un sistema de riego automático.

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

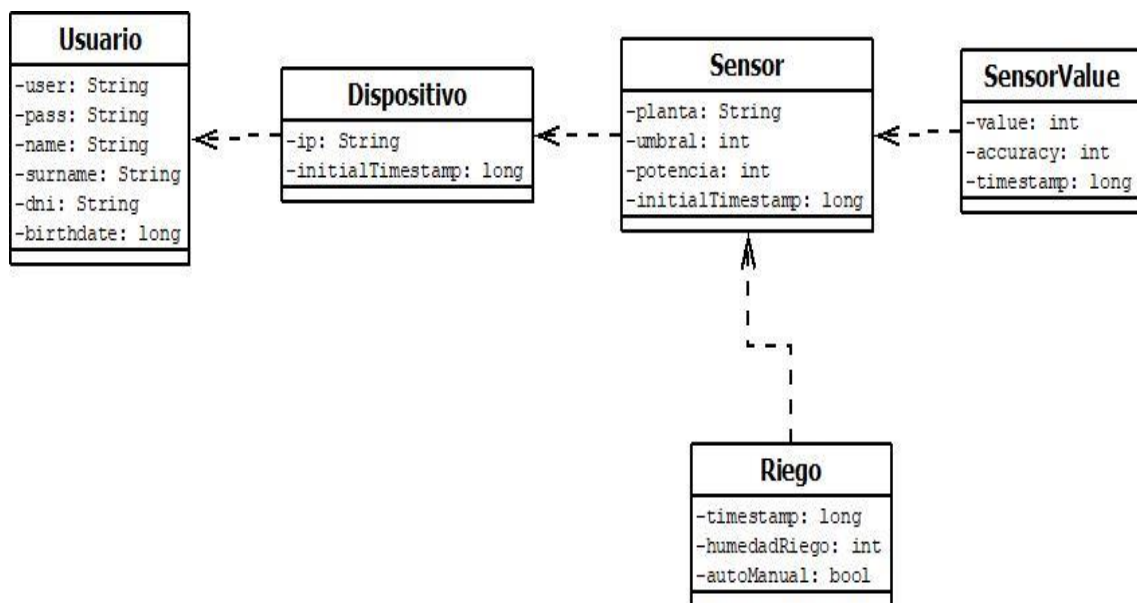


Resistencias

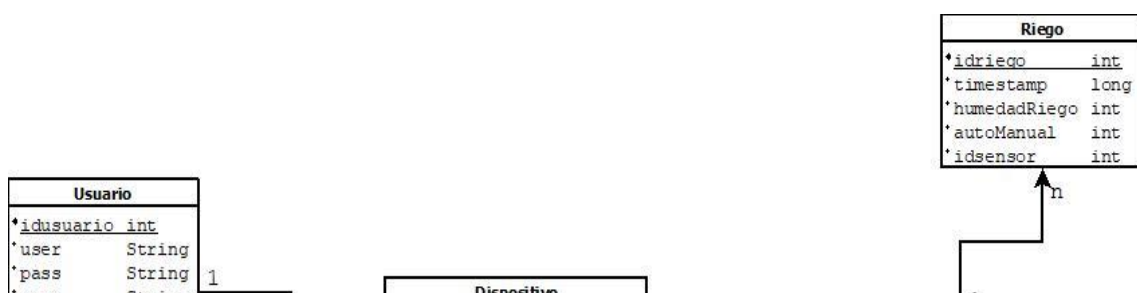
Relé 3,3 V

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.

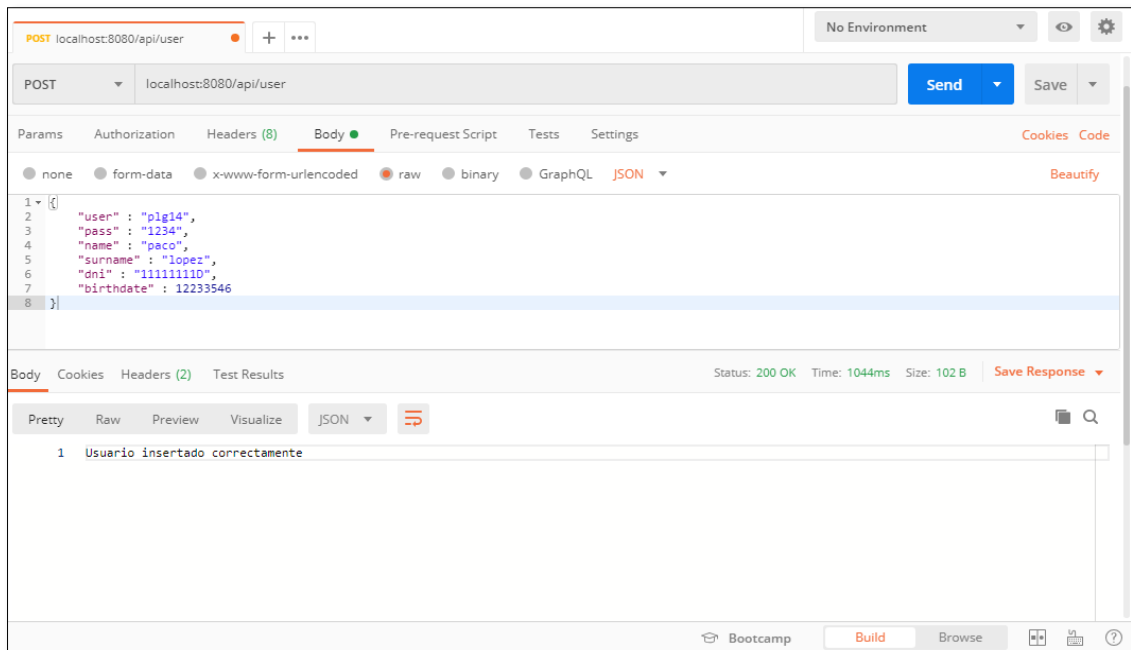
■ "/api/user"

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).

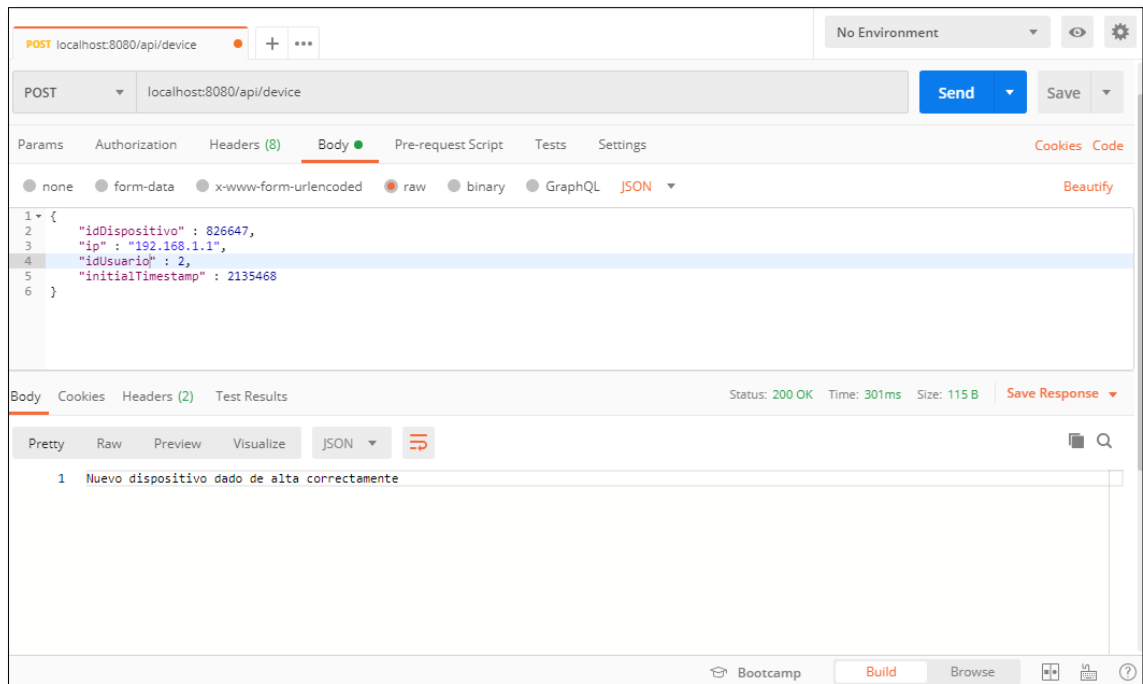
El cuerpo es el siguiente:

```
{
  "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"

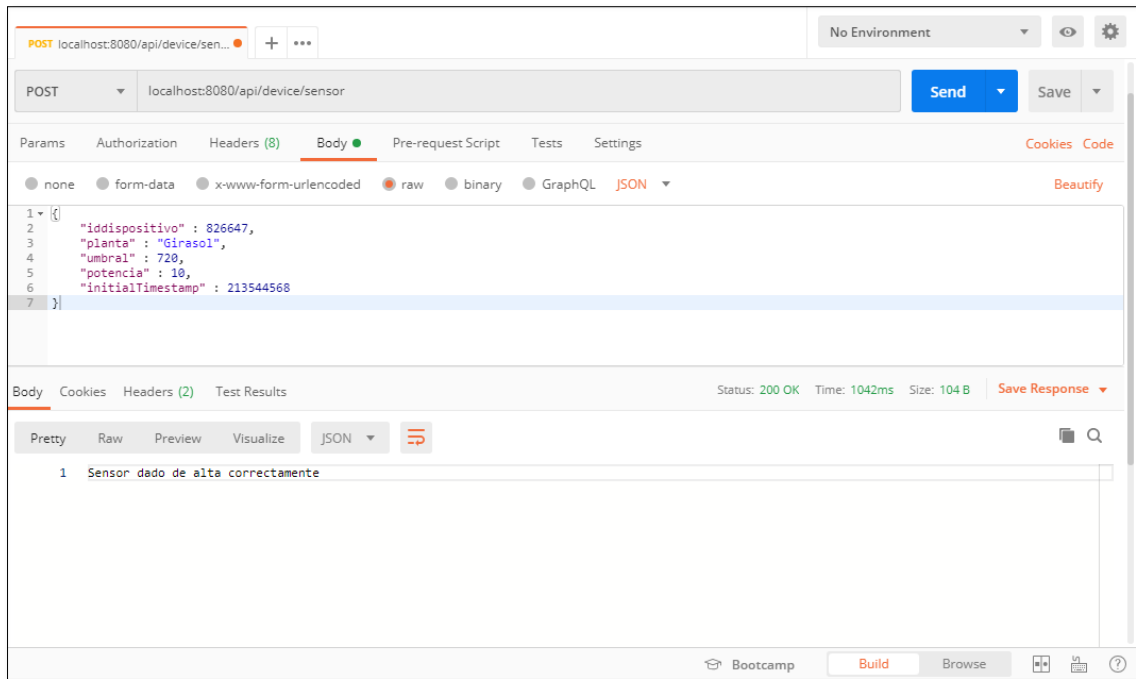
Cuando el usuario conecte un sensor al dispositivo, dicho usuario dará de alta al sensor.

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.



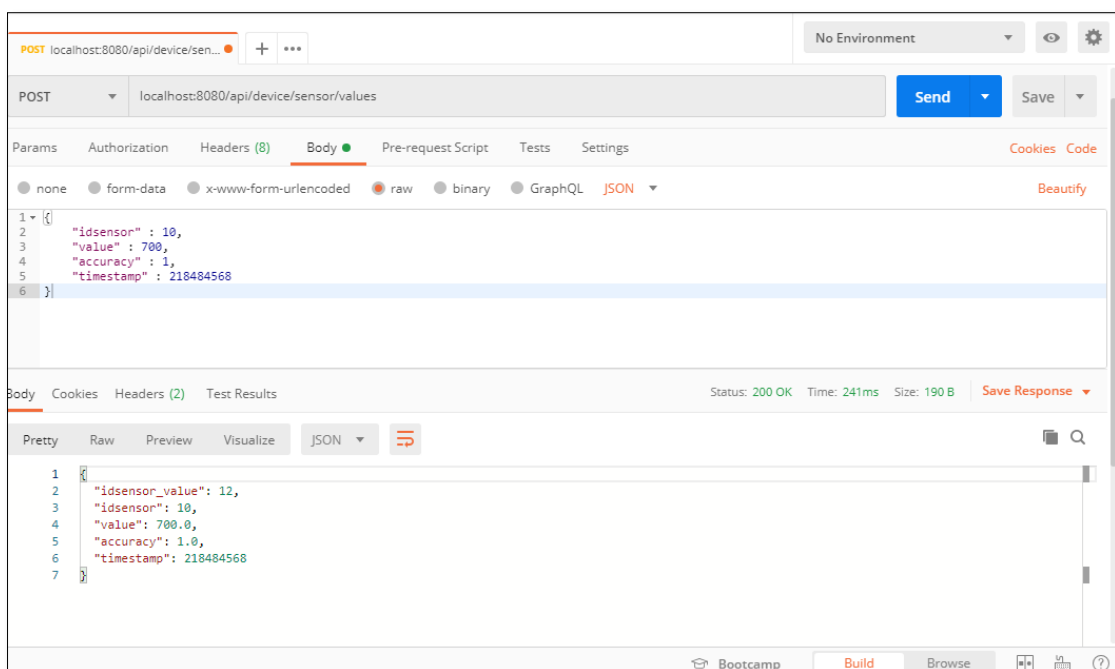
■ "/api/device/sensor/values"

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"

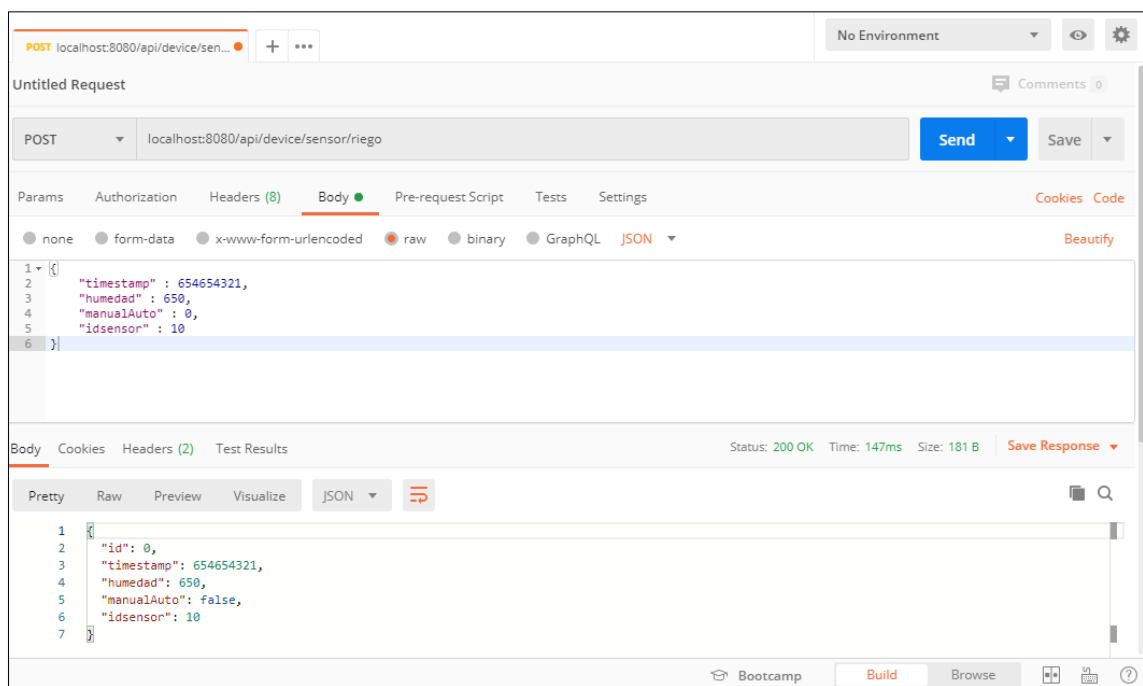
Este método se llamará cada vez que se realice la acción de riego, para guardar un historial de los riegos.

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.



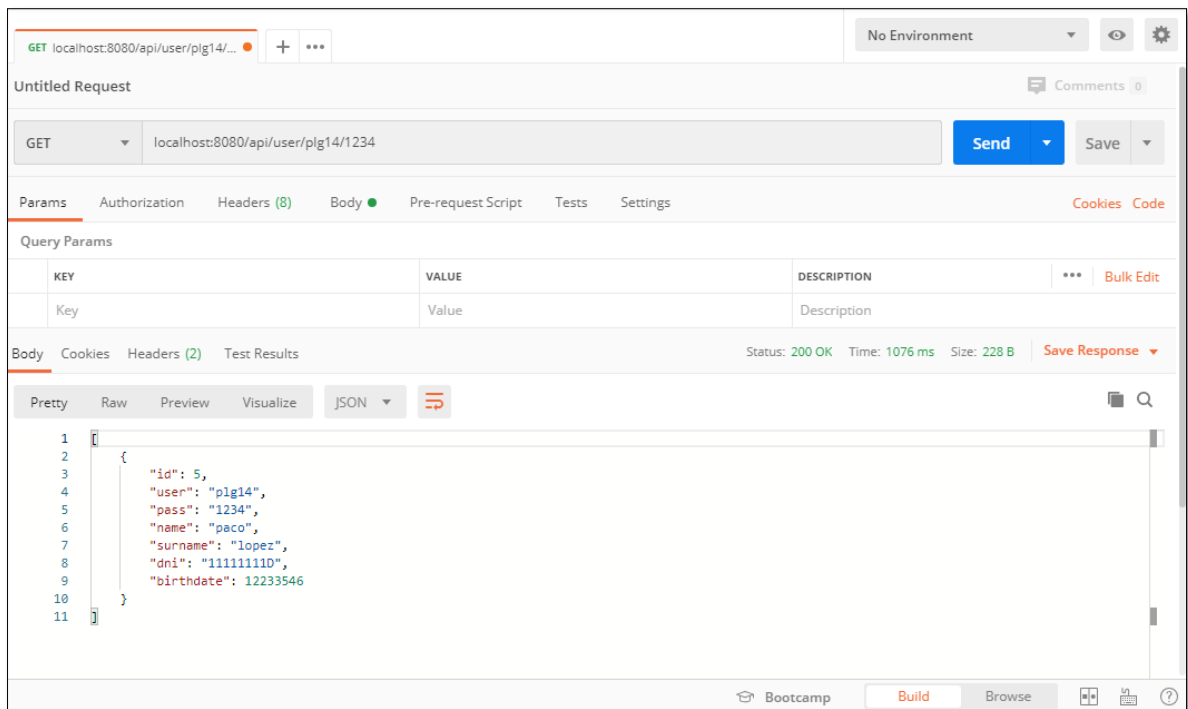
3.2.- GET

Usaremos los GET para hacer SELECT a la BBDD.

■ "/api/user/:user/:pass"

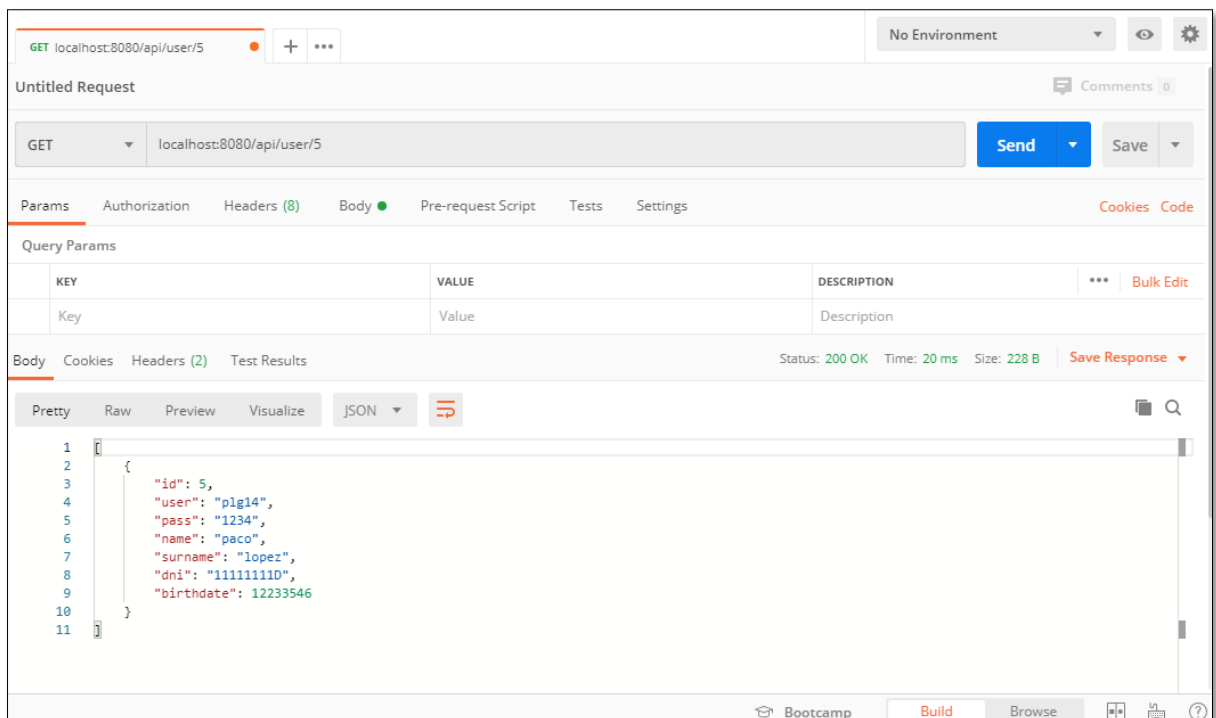
Esta URL se usará para el login del usuario. Se pedirá el usuario y la contraseña, rellena en el formulario de login. Si es correcto, se accederá a la aplicación, si es incorrecto, se enviará un mensaje de error.

Como es un GET, no hay cuerpo.



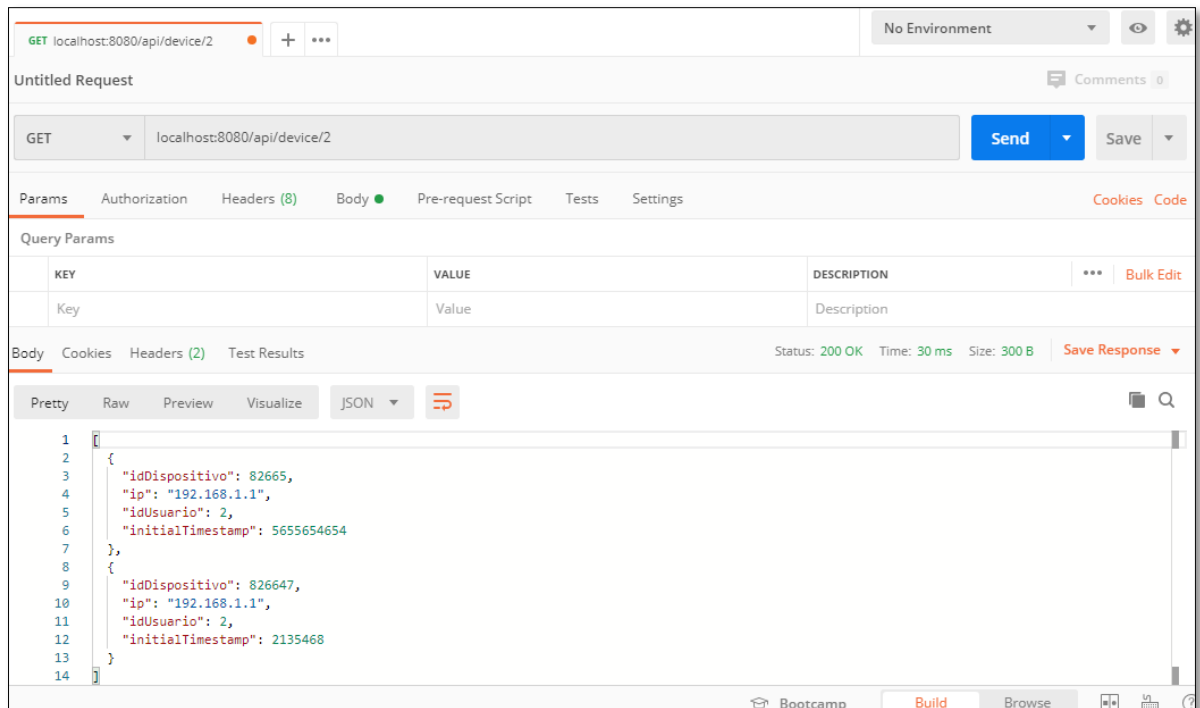
■ `"/api/user/:idusuario"`

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.



■ "/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

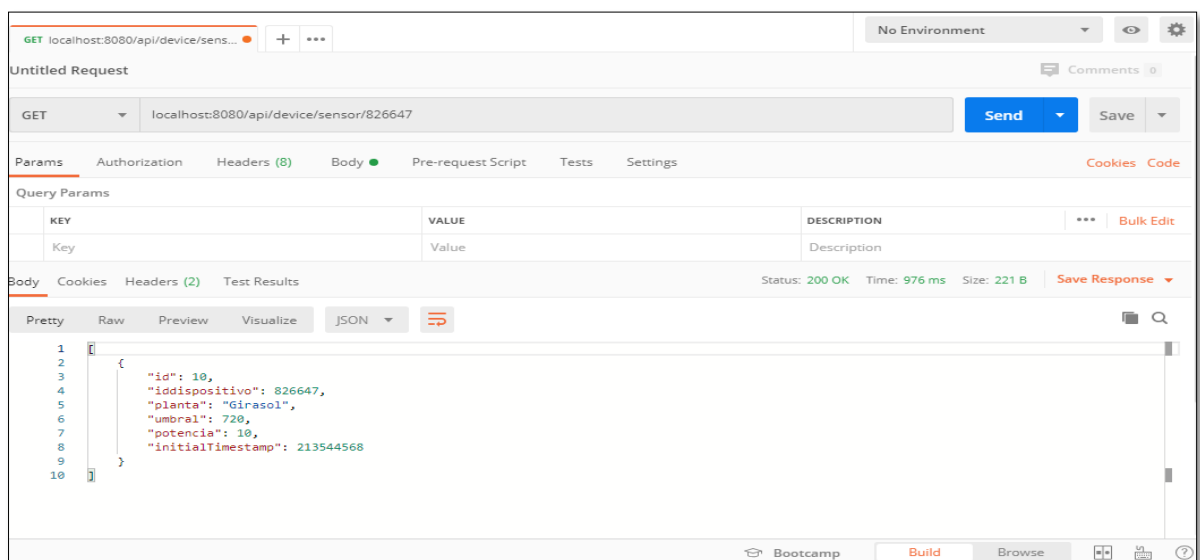
Pretty Raw Preview Visualize JSON

```
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/sensor/826647

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

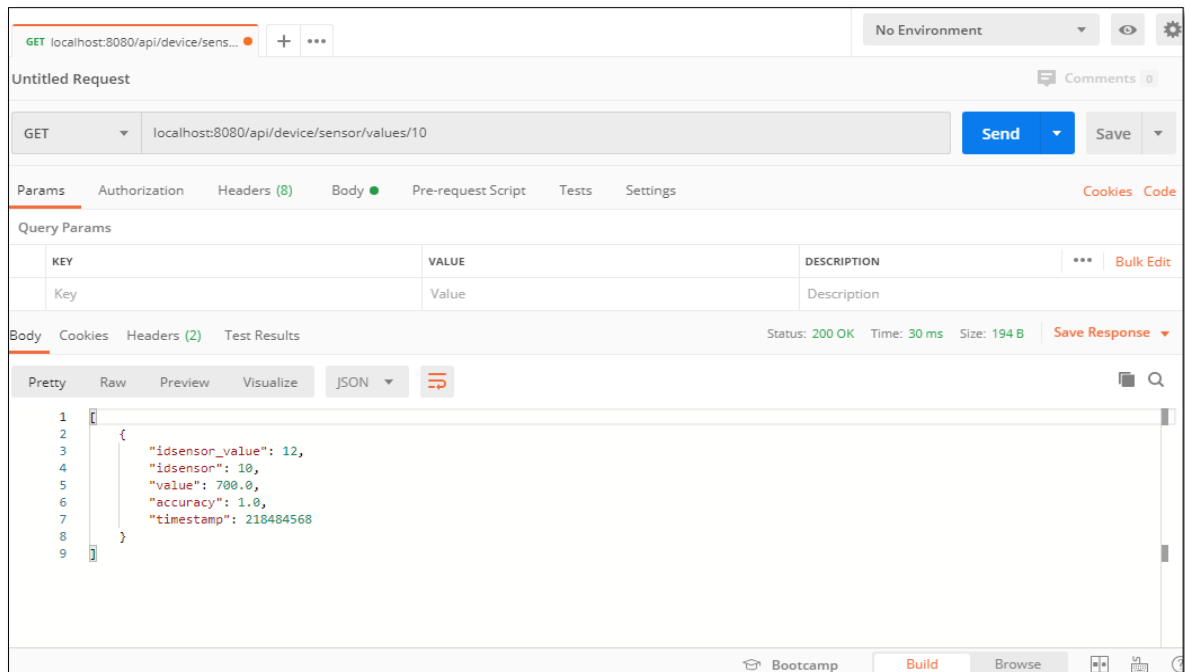
Pretty Raw Preview Visualize JSON

```
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

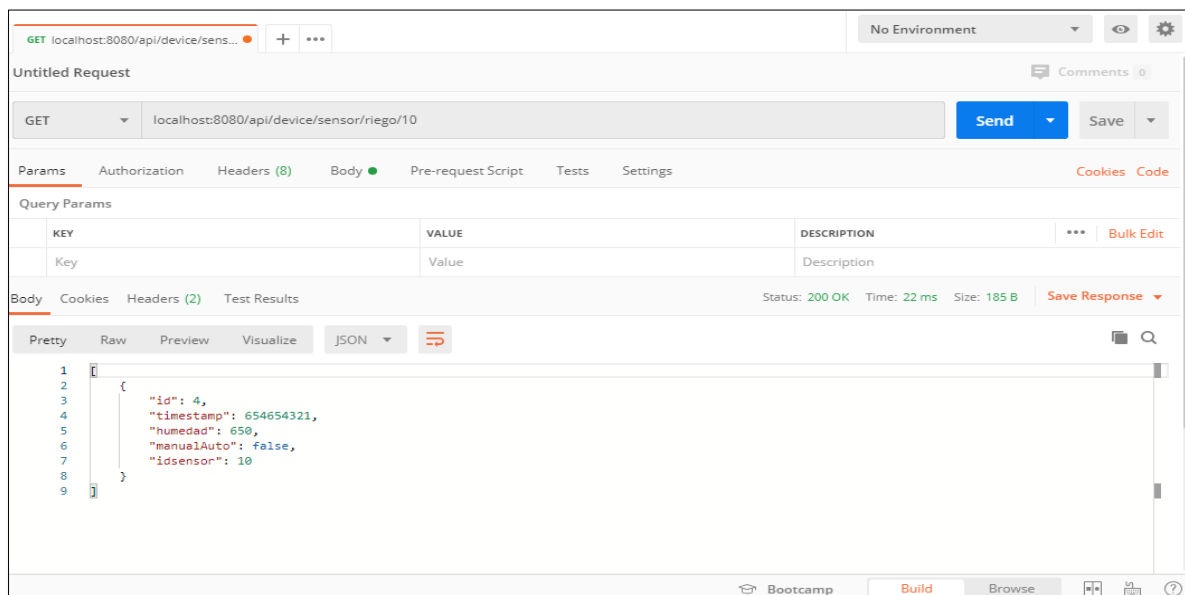
■ "/api/device/sensor/values/:idSensor"

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



■ "/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.



3.3.- PUT

Usaremos los PUT para hacer UPDATE en la BBDD.

■ `"/api/user/:idusuario"`

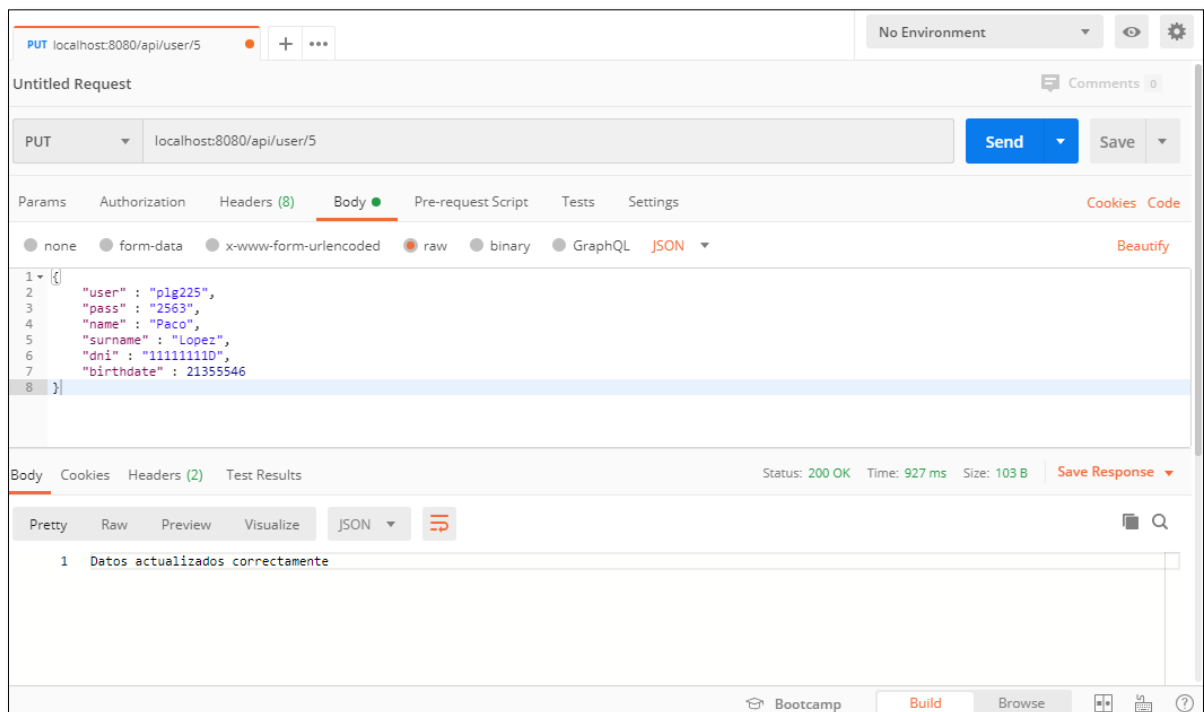
Dado el id del usuario, se actualizarán los datos de dicho usuario con los datos rellenos en un formulario.

El cuerpo es el siguiente:

```
{
  "user" : "plg225",
  "pass" : "2563",
  "name" : "Paco",
  "surname" : "Lopez",
  "dni" : "11111111D",
  "birthdate" : 2135546
}
```

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.



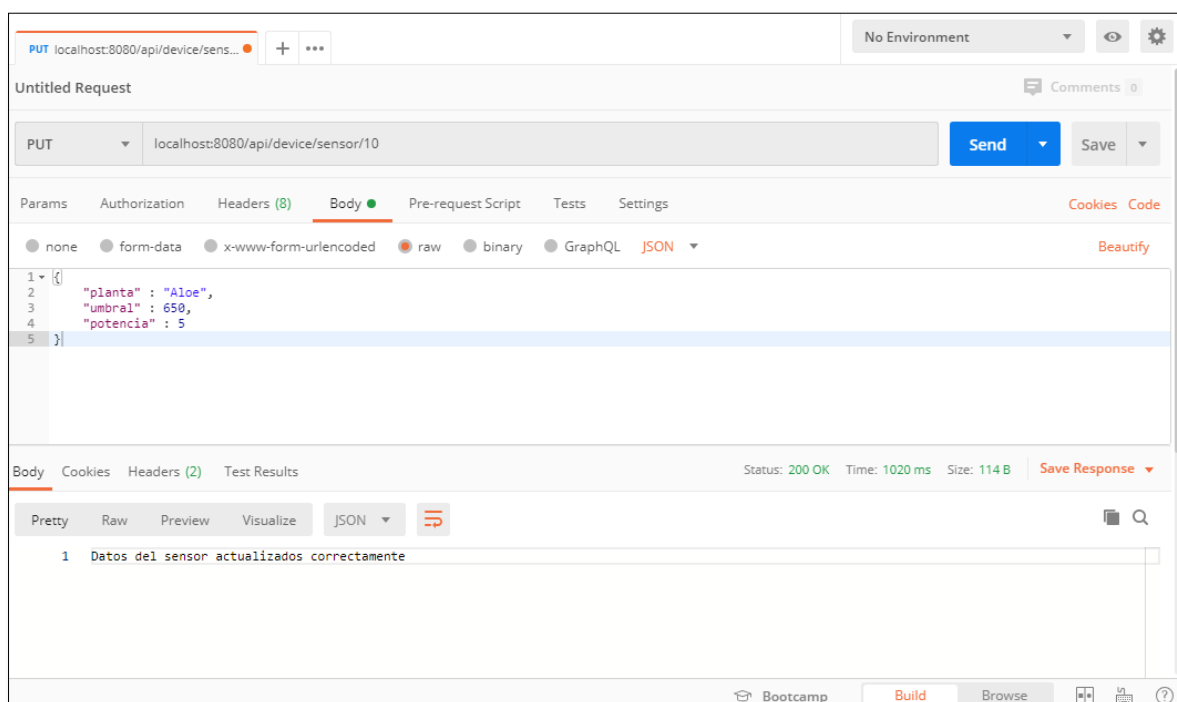
■ "/api/device/sensor/:idsensor"

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.



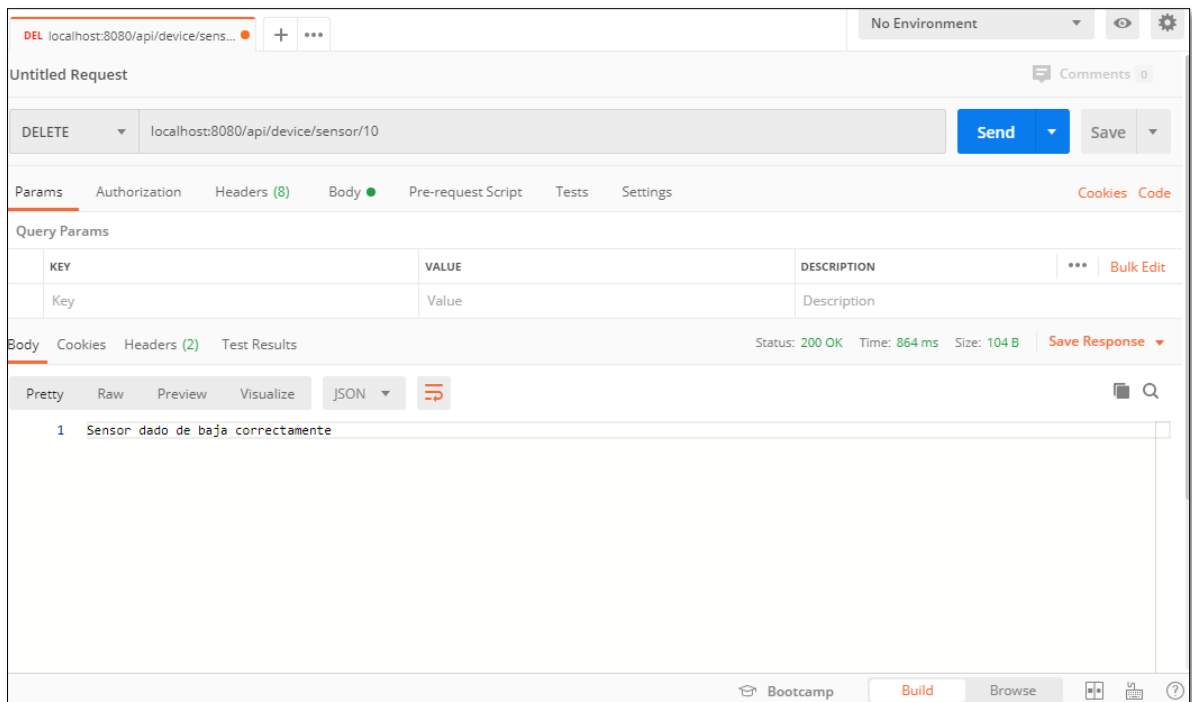
3.4.- DELETE

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.

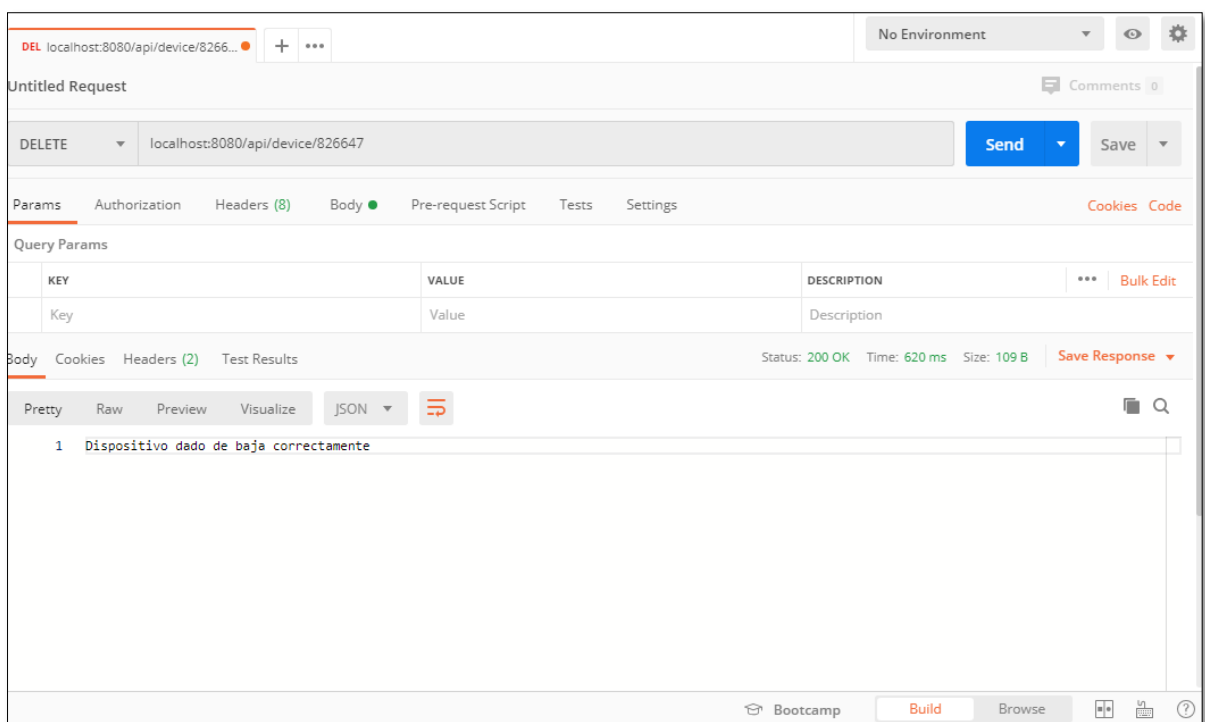
Como sólo nos interesa el id, no habrá cuerpo. Un mensaje al cliente sería que el sensor se ha dado de baja correctamente.



■ "/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.

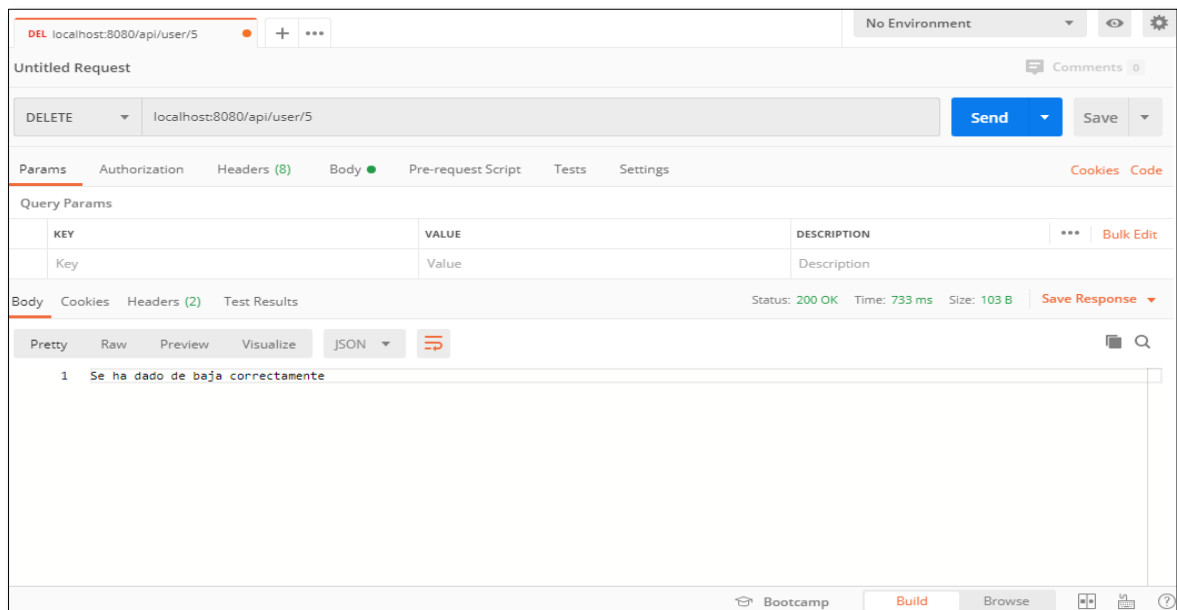
Como sólo nos interesa el id, no habrá cuerpo. Un mensaje al cliente sería que el dispositivo se ha dado de baja correctamente.



■ "/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.



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 leído 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 valor del
    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; //aquí 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");
}

```

6.- ENTREGA FINAL

```
#include <Arduino.h>
#include "ArduinoJson.h"
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <SoftwareSerial.h>
#include "ESPDateTime.h"

char responseBuffer[300];
WiFiClient client;

const int pinSensor = A0;
const int pinBoton = 14;
const int ledRojo = 0;
const int ledAmarillo = 4;
const int ledVerde = 5;
const int bomba = 2;

String SSID = "Dunar"; //nombre de la red a la que el ESP se conectara
String PASS = "d7?a35D9EnaPepXY?c!4"; //contraseña de acceso

String SERVER_IP = "192.168.1.104"; //poner direccion ip
int SERVER_PORT = 8080; //poner puerto de escucha

int device = 82664; //poner aquí el id del dispositivos (teoricamente
guardado en flash)
bool boton = false;

void guardarLecturas(int valor); //insert en la bbdd de las lecturas
void regar(boolean pulsador, int valor, int potencia); //insert en la
bbdd de los riegos
int getUmbral(); //get del umbral para saber cuando regar
void estadoLEDs(int valor, int umbral);
int getPotencia();
void riegoManual(int valor, int potencia);

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

    pinMode(pinSensor, INPUT);
    pinMode(pinBoton, INPUT);

    pinMode(ledRojo, OUTPUT);
    pinMode(ledAmarillo, OUTPUT);
    pinMode(ledVerde, OUTPUT);

    pinMode(bomba, OUTPUT);

    digitalWrite(ledRojo, LOW);
    digitalWrite(ledAmarillo, LOW);
    digitalWrite(ledVerde, LOW);
    digitalWrite(bomba, LOW);

    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() {
    int umbral = getUmbral();
    int potencia = getPotencia();
    int valor = analogRead(pinSensor);
    estadoLEDS(valor, umbral);
    if(valor < 1024) {
        guardarLecturas(valor);
        if(digitalRead(pinBoton)) {
            riegoManual(valor, potencia);
        } else {
            if(valor >= umbral) {
                regar(false, valor, potencia);
            }
        }
    }
    delay(10000);
}

void riegoManual(int valor, int potencia) {
    if(boton) {
        regar(true, valor, potencia);
    }
}

int getUmbral() {
    int umbral = -1;
    if(WiFi.status() == WL_CONNECTED) {
        HTTPClient http;
        http.begin(client, SERVER_IP, SERVER_PORT,
"/api/device/sensor/id/"+String(pinSensor), 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 umbral: ");
            Serial.println(error.c_str());
        }
        umbral = doc[0]["umbral"].as<int>();
    }
    return umbral;
}

int getPotencia() {
    int potencia = -1;
    if(WiFi.status() == WL_CONNECTED) {
        HTTPClient http;

```

```

    http.begin(client, SERVER_IP, SERVER_PORT,
"/api/device/sensor/id/"+String(pinSensor), 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 potencia: ");
        Serial.println(error.c_str());
    }
    potencia = doc[0]["potencia"].as<int>();
}

return potencia;
}

void guardarLecturas(int valor){
    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"] = pinSensor;
        doc["value"] = valor;
        doc["accuracy"] = 1;
        doc["timestamp"] = DateTime.now();

        String output;

        serializeJson(doc, output);

        http.POST(output);

        String payload = http.getString();
    }
}

void regar(boolean pulsador, int valor, int potencia){
    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"] = DateTime.now();

```

```

    doc["humedad"] = valor; //valor leido por el sensor
    doc["manualAuto"] = pulsador; //estado del boton
    doc["idsensor"] = pinSensor;

    String output;
    serializeJson(doc, output);

    http.POST(output);

    int tiempo = 30 * potencia;

    digitalWrite(bomba,HIGH);
    delay(tiempo);
    digitalWrite(bomba,LOW);

    String payload = http.getString();
  }
}

void estadoLEDs(int valor, int umbral){
  double fv = (umbral/1.2);
  if(valor >= 1024){
    digitalWrite(ledRojo,HIGH);
    digitalWrite(ledAmarillo,LOW);
    digitalWrite(ledVerde,LOW);
    boton = false;
  }else if(valor>=fv){
    digitalWrite(ledRojo,LOW);
    digitalWrite(ledAmarillo,HIGH);
    digitalWrite(ledVerde,LOW);
    boton = true;
  }else if(valor<fv){
    digitalWrite(ledRojo,LOW);
    digitalWrite(ledAmarillo,LOW);
    digitalWrite(ledVerde,HIGH);
    boton = false;
  }
}
}

```

6.1.- Variables Globales:

```

const int pinSensor = A0;
const int pinBoton = 14;
const int ledRojo = 0;
const int ledAmarillo = 4;
const int ledVerde = 5;
const int bomba = 2;

```

Estas son las variables que usaremos para el funcionamiento del proyecto:

- **pinSensor:** indica el pin en donde esta conectado el sensor de humedad, este es un pin analogico.
- **pinBoton:** pin en donde conectaremos un botón para hacer un riego manual.
- **ledRojo:** pin para controlar un led rojo.
- **ledAmarillo:** pin para controlar un led amarillo.
- **ledVerde:** pin para controlar un verde.
- **bomba:** pin para controlar el motor de agua.

```
String SSID = "Dunar";
```

```
String PASS = "d7?a35D9EnaPepXY?c!4";

String SERVER_IP = "192.168.1.104"; //poner direccion ip
int SERVER_PORT = 8080; //poner puerto de escucha
```

Estas son variables para la conexión del wifi de la placa Esp8266.

```
int device = 82664;
bool boton = false;
```

La variable device indica el id del dispositivo, este dato indica que sensores están asociados a un usuario de la base datos.

La variable botón será una variable que funciona como cerrojo, cuando sea true eso indica que es posible hacer un riego manual y cuando está puesto a false el riego manual se desactiva.

6.2.- Funciones Auxiliares:

Las funciones que usaremos son las siguientes:

```
void guardarLecturas(int valor);
void regar(boolean pulsador, int valor, int potencia);
int getUmbral();
void estadoLEDs(int valor, int umbral);
int getPotencia();
void riegoManual(int valor, int potencia);
```

6.2.1.- Función guardarLecturas():

La función **guardarLecturas** se encarga de hacer inserciones de las lecturas del sensor. El valor leído por el sensor se envía como parámetro en la variable valor. Con este valor construimos una variable JSON, después hacemos un POST de la variable JSON mediante la API REST. La función de la API REST se encargará de hacer la inserción de ese JSON en la base de datos.

```
void guardarLecturas(int valor){
    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"] = pinSensor;
        doc["value"] = valor;
        doc["accuracy"] = 1;
        doc["timestamp"] = DateTime.now();

        String output;

        serializeJson(doc, output);

        http.POST(output);
    }
}
```

```

        String payload = http.getString();
    }
}

```

6.2.2.- Función regar():

La función **regar** hace que el motor se active cuando esta función sea llamada y también hace una inserción en la tabla de riego de la base de datos indicando el tiempo, el valor actual del sensor, si es un riego manual o automático, y el id del sensor que hizo la lectura. Esta función recibe 3 parámetros:

boolean pulsador: indica el tipo de riego, true si es un riego manual o false si es automático.

int valor: valor del sensor de humedad cuando se llama esta función.

int potencia: indica cuando tiempo va a funcionar el motor (máximo 1)

Con estos parámetros y junto con un timestamp que creamos con la función Date de Arduino, debemos crear un JSON que insertaremos en la base de datos. La inserción se hace mediante un POST de la API REST.

A continuación, debemos encender el motor durante el tiempo que indique la variable potencia, para encender el motor debemos escribir HIGH en el pin del motor. El tiempo que estará encendido se calcula multiplicándolo por 30 y es controlado por un delay como podremos ver en el siguiente código. Después del delay pondremos el pin del motor en LOW para hacer que el motor se apague.

```

void regar(boolean pulsador, int valor, int potencia){
    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"] = DateTime.now();
        doc["humedad"] = valor; //valor leído por el sensor
        doc["manualAuto"] = pulsador; //estado del boton
        doc["idsensor"] = pinSensor;

        String output;
        serializeJson(doc, output);

        http.POST(output);

        int tiempo = 30 * potencia;

        digitalWrite(bomba,HIGH);
        delay(tiempo);
        digitalWrite(bomba,LOW);

        String payload = http.getString();
    }
}

```

6.2.3.- Función getUmbral():

La función **getUmbral** nos devuelve el valor del campo umbral de la tabla de sensor de la base de datos. Mediante la API REST hacemos una consulta a la base de datos del valor del umbral del sensor que hayamos indicado en la variable pinSensor.

```
int getUmbral(){
    int umbral = -1;
    if(WiFi.status() == WL_CONNECTED){
        HTTPClient http;
        http.begin(client, SERVER_IP, SERVER_PORT,
"/api/device/sensor/id/"+String(pinSensor), 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 umbral: ");
            Serial.println(error.c_str());
        }
        umbral = doc[0]["umbral"].as<int>();
    }
    return umbral;
}
```

6.2.4- Función estadoLEDs():

La función **estadoLEDs** se encarga del control de los leds. Esta función recibe 2 parámetros:

- **int valor:** parámetro que indica la humedad actual.
- **int umbral:** indica cuando es posible hacer un riego.

Los LEDs nos sirven como indicación para ver el estado actual de la humedad:

El led rojo se encenderá cuanto el sensor de humedad no está bien conectado. Cuando el sensor de humedad no está conectado a tierra o no está conectado a la tierra nos da una lectura de 1024.

El led amarillo se enciende cuando el valor del sensor supera al umbral, esto indica que es necesario hacer un riego, durante este tiempo se activará la posibilidad de hacer un riego manual (la variable botón pasa a true) si pulsamos el botón si no lo presionamos el riego se hará automáticamente.

El led verde se enciende cuando el valor del sensor es inferior al umbral, esto indica que la humedad es correcta y no es necesario hacer un riego.

```
void estadoLEDs(int valor, int umbral){
    double fv = (umbral/1.2);
    if(valor >= 1024){
        digitalWrite(ledRojo,HIGH);
        digitalWrite(ledAmarillo,LOW);
        digitalWrite(ledVerde,LOW);
    }
```

```

    boton = false;
} else if (valor >= fv) {
    digitalWrite(ledRojo, LOW);
    digitalWrite(ledAmarillo, HIGH);
    digitalWrite(ledVerde, LOW);
    boton = true;
} else if (valor < fv) {
    digitalWrite(ledRojo, LOW);
    digitalWrite(ledAmarillo, LOW);
    digitalWrite(ledVerde, HIGH);
    boton = false;
}
}

```

6.2.5.- Función getPotencia():

La función getPotencia nos devuelve el valor del campo potencia de la tabla de sensor de la base de datos. Su funcionamiento es similar al de la función getUmbral.

```

int getPotencia() {
    int potencia = -1;
    if (WiFi.status() == WL_CONNECTED) {
        HTTPClient http;
        http.begin(client, SERVER_IP, SERVER_PORT,
"/api/device/sensor/id/" + String(pinSensor), 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 potencia: ");
            Serial.println(error.c_str());
        }
        potencia = doc[0]["potencia"].as<int>();
    }
    return potencia;
}

```

6.2.6.- Función riegoManual():

La función riegoManual es una función auxiliar que en caso de que el riego manual este activado (esto esta indicado por el valor de la variable botón) hará una llamada a la función regar, a esta función le pasamos como primer parámetro true para indicar que se trata de un riego manual.

```

void riegoManual(int valor, int potencia) {
    if (boton) {
        regar(true, valor, potencia);
    }
}

```

6.3.- Función Void Setup():

Dentro de esta función ira la inicialización de los diferentes pines y del monitor serie, además también se hace la conexión a la red WIFI

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

  pinMode(pinSensor, INPUT);
  pinMode(pinBoton, INPUT);

  pinMode(ledRojo, OUTPUT);
  pinMode(ledAmarillo, OUTPUT);
  pinMode(ledVerde, OUTPUT);

  pinMode(bomba, OUTPUT);

  digitalWrite(ledRojo, LOW);
  digitalWrite(ledAmarillo, LOW);
  digitalWrite(ledVerde, LOW);
  digitalWrite(bomba, LOW);

  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());
}
```

6.4.- Función Void Loop():

Esta será la función que se ejecutará constantemente, en cada iteración haremos la lectura del pin del sensor mediante un analogRead y lo almacenamos en la variable valor, también llamaremos a las funciones getUmbral y getPotencia para consultar sus valores y almacenarlos en las variables umbral y potencia respectivamente.

A continuación si valor es menor a 1024 almacenamos su valor en la base de datos mediante la función guardarLecturas(). Si el botón de riego manual es pulsado llamaremos a la función de riego manual, si no es pulsado se hace un riego automático llamando a la función regar con su primer parámetro de entrada como false.

Al final tendremos un delay de 10 segundos.

```
void loop() {
  int umbral = getUmbral();
  int potencia = getPotencia();
  int valor = analogRead(pinSensor);
  estadoLEDs(valor, umbral);
  if(valor < 1024) {
    guardarLecturas(valor);
    if(digitalRead(pinBoton)) {
      riegoManual(valor, potencia);
    }
  }
  delay(10000);
}
```



```
    }else{  
        if(valor >= umbral){  
            regar(false, valor, potencia);  
        }  
    }  
    }  
    delay(10000);  
}
```