

TALLER 5. Protocolo REST

Inicio

El objetivo de este taller es entender el funcionamiento del mecanismo de transmisión REST, basado en el protocolo HTTP, a través de la conexión a diferentes servidores REST y desde diversos clientes.

Se trabajará sobre el proyecto grupal que venimos trabajando en los talleres anteriores, realizando el envío de los datos en el formato JSON definido.

Se trabajará en 2 ambientes diferentes:

1. Servidor local - json-server
2. Plataforma IoT que implemente el protocolo Rest – Ubidots.

Procedimiento – Servidor Local

1. **Json-server.** Es una utilidad que permite crear un servidor Rest (api rest) sin necesidad de código en muy corto tiempo y sirve para realizar pruebas de clientes Rest. Está basado en JavaScript y corre sobre el servidor Node.js. Por tanto, lo primero que se debe hacer es instalar Node.js.

Para la instalación se debe descargar el software desde la página oficial de Node.js: <https://nodejs.org/es/>

Una vez descargado el archivo procedemos con la instalación, que no es más que seguir el wizard y dar Next. Junto con Node.js se instalará el gestor de paquetes NPM, que es una herramienta que facilita la instalación de paquetes necesarios para nuestros proyectos.

Para que Node.js se pueda ejecutar desde cualquier sitio de nuestro equipo puede ser necesario agregarlo a las variables de entorno.

Creamos una carpeta para almacenar nuestro proyecto, en este caso la he llamado protocoloREST y la he creado en el escritorio. Dentro de la carpeta ejecutamos el comando de instalación de json-server: `npm i json-server`

```

D:\Users\zsolarte\Desktop\protocoloREST>npm i json-server
npm WARN saveError ENOENT: no such file or directory, open 'D:\Users\zsolarte\Desktop\protocoloREST\package.json'
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN enoent ENOENT: no such file or directory, open 'D:\Users\zsolarte\Desktop\protocoloREST\package.json'
npm WARN protocoloREST No description
npm WARN protocoloREST No repository field.
npm WARN protocoloREST No README data
npm WARN protocoloREST No license field.

+ json-server@0.17.0
added 182 packages from 85 contributors and audited 182 packages in 59.762s

20 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

```

Dentro de la misma carpeta creamos un archivo denominado `datos.json` que contendrá los datos con la estructura que requerimos para hacer las pruebas del servidor Rest. El contenido de dicho archivo puede ser como se muestra a continuación:

```

{
  "datos": [
    {
      "id": 1
    }
  ]
}

```

Cuando ya tenemos el archivo json, podemos proceder a lanzar el servidor, lo cual se hace con el siguiente comando: `json-server --watch datos.json`

```

D:\Users\zsolarte\Desktop\protocoloREST>json-server --watch datos.json

\{^_^}/ hi!

Loading datos.json
Done

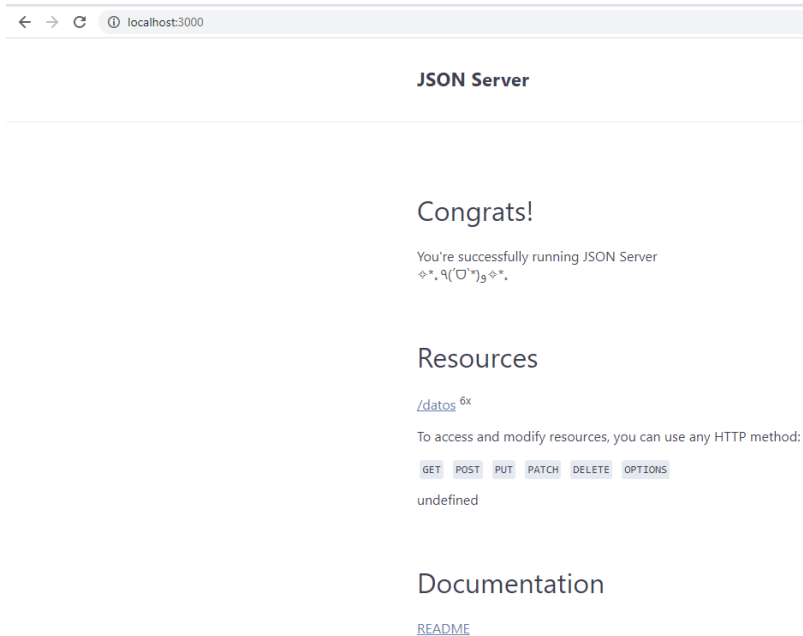
Resources
http://localhost:3000/datos

Home
http://localhost:3000

Type s + enter at any time to create a snapshot of the database
Watching...

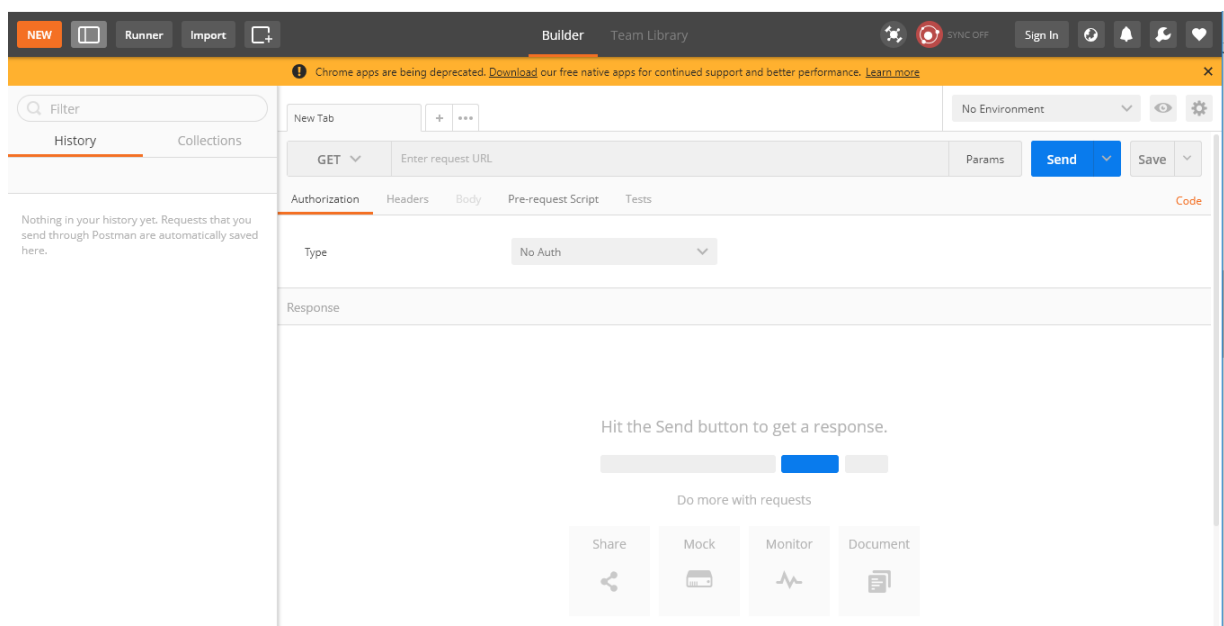
```

Si ingresamos a la URL de Home podremos ver las opciones que se tienen para acceder al servidor montado:



Como se puede observar la ruta para acceder a los recursos será <http://localhost:3000/datos> y se podrá usar cualquiera de los métodos http.

2. **Ciente Postman.** Postman es una extensión del navegador Google Chrome, que permite el envío de peticiones HTTP REST sin necesidad de desarrollar un cliente. Permite modificar todos los parámetros de la petición. Así, se puede editar el header de la petición. También se puede editar el contenido, y añadir campos tanto en Json, XML o texto plano, entre otros. Lo primero que debemos hacer es instalarlo.



Probamos los diferentes métodos HTTP para acceder al servidor,

POST

POST

http://localhost:3000/datos/

Send

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1

2

3

4

"temperatura":24,

"humedad":50

Body

Cookies

Headers (14)

Test Results

Status: 201 Created

Time: 520 ms

Size: 535 B

Save

Pretty

Raw

Preview

Visualize

JSON

1

2

3

4

5

"temperatura": 24,

"humedad": 50,

"id": 3

GET

GET

http://localhost:3000/datos/

Send

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body

Cookies

Headers (12)

Test Results

Status: 200 OK

Time: 514 ms

Size: 522 B

Save

Pretty

Raw

Preview

Visualize

JSON

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

{

"id": 1

},

{

"temperatura": 24,

"humedad": 50,

"id": 2

},

{

"temperatura": 24,

"humedad": 50,

"id": 3

}

PUT

PUT

http://localhost:3000/datos/1

Send

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1

2

3

4

"temperatura":30,

"humedad":20

Body

Cookies

Headers (12)

Test Results

Status: 200 OK

Time: 525 ms

Size: 423 B

Save

Pretty

Raw

Preview

Visualize

JSON

1

2

3

4

5

"temperatura": 30,

"humedad": 20,

"id": 1

DELETE

DELETE

localhost:3000/datos/6

Send

Body

Cookies

Headers (12)

Test Results

Status: 200 OK

Time: 511 ms

Size: 372 B

Save

Pretty

Raw

Preview

Visualize

3. **Plataforma hardware.** En nuestro hardware (plataforma+ sensores) incluir las librerías y el código que permite la conexión al servidor REST.

Acondicionamos el servidor para que reciba datos desde un dispositivo externo, se debe lanzar con la dirección IP del equipo donde se está ejecutando el json server:

```
D:\Users\zsolarte\Desktop\protocoloREST>json-server --host 192.168.43.227 --watch datos.json

\{^_^}/ hi!

Loading datos.json
Done

Resources
http://192.168.43.227:3000/datos

Home
http://192.168.43.227:3000

Type s + enter at any time to create a snapshot of the database
Watching...
```

Para esta práctica trabajaremos con otra plataforma hardware, el ESP32. Para poder trabajar con ella debemos seguir los pasos que aparecen en el siguiente enlace: <https://www.taloselectronics.com/blogs/tutoriales/programar-esp32-con-ide-arduino>.

Ahora, creamos el sketch. Para el ESP32 debemos usar la librería httpclient, que él ya la tiene por defecto.

ESP32 – POST

```
#include <HTTPClient.h>
#include <WiFi.h>
#include <ArduinoJson.h>

const char* ssid = "*****"; //El SSID de la red wifi a la que se conectará
const char* password = "*****"; //El password para conectarse a la red
inalambrica

void setup() {
  Serial.begin(115200); //Serial connection
  setup_wifi(); //WiFi connection
  delay(1500);
}

void setup_wifi() {

  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);
```

```

while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}

Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
}

void loop() {

    int temp = analogRead(34); //lectura de los sensores en los pines 34 y 35
    int hum = analogRead(35);

    String variable;

    DynamicJsonDocument doc(1024); //creacion del json

    doc["temperatura"] = temp;
    doc["humedad"] = hum;

    serializeJson(doc, variable);
    Serial.println("dato a enviar: "+ variable);

    HTTPClient http;    //Declare object of class HTTPClient
    WiFiClient client;
    //Specify request destination
    http.begin(client, "http://192.168.43.227:3000/datos/");
    http.addHeader("Content-Type", "application/json"); //Specify content-
type header

    int httpCode = http.POST(variable); //Send the request
    String payload = http.getString(); //Get the response payload

    Serial.println(httpCode); //Print HTTP return code
    Serial.println(payload); //Print request response payload

    http.end(); //Close connection

    delay(5000); //Send a request every 5 seconds
}

```

Al ejecutar el sketch se ven los datos enviados:

```
}
dato a enviar: {"temperatura":288,"humedad":4095}
201
{
  "temperatura": 288,
  "humedad": 4095,
  "id": 19
}
dato a enviar: {"temperatura":278,"humedad":4095}
201
{
  "temperatura": 278,
  "humedad": 4095,
  "id": 20
}
}
```

☒ Autoscroll ☐ Mostrar marca temporal

Nueva línea Nueva línea 115200 baudio Limpiar salida

Y se observa que se reciben en el servidor:

```
Simbolo del sistema - json-server --host 192.168.43.227 --watch datos.json
D:\Users\zsolarte\Desktop\protocoloREST>json-server --host 192.168.43.227 --watch datos.json

\{^_^}/ hi!

Loading datos.json
Done

Resources
http://192.168.43.227:3000/datos

Home
http://192.168.43.227:3000

Type s + enter at any time to create a snapshot of the database
Watching...

GET /datos 200 10.681 ms - 726
POST /datos/ 201 168.526 ms - 54
POST /datos/ 201 50.902 ms - 55
POST /datos/ 201 51.847 ms - 55
POST /datos/ 201 49.180 ms - 55
POST /datos/ 201 81.227 ms - 55
POST /datos/ 201 53.271 ms - 56
POST /datos/ 201 49.132 ms - 56
POST /datos/ 201 52.735 ms - 56
POST /datos/ 201 51.364 ms - 55
POST /datos/ 201 147.175 ms - 55
POST /datos/ 201 57.386 ms - 55
POST /datos/ 201 52.867 ms - 55
```

También se pueden observar en Postman:

GET http://192.168.43.227:3000/datos/ Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (13) Test Results

Status: 200 OK Time: 514 ms Size: 1.89 KB Sav

Pretty Raw Preview Visualize JSON

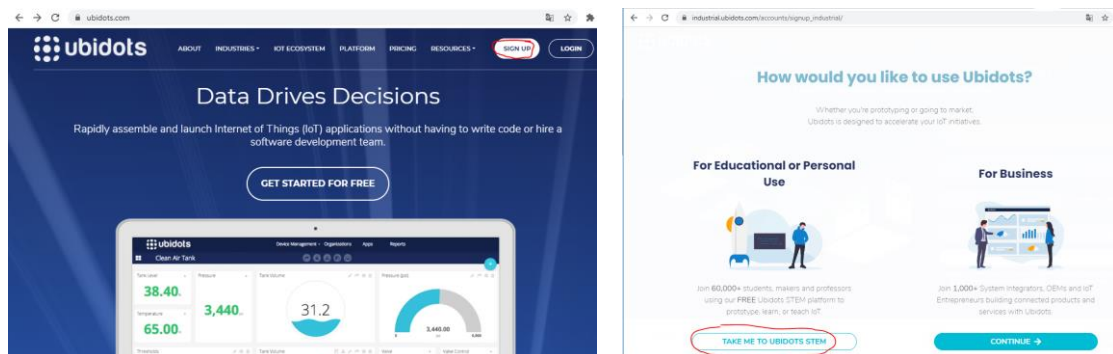
```
84  {
85    "humedad": 4095,
86    "id": 15
87  },
88  {
89    "temperatura": 4095,
90    "humedad": 4095,
91    "id": 16
92  },
93  {
94    "temperatura": 287,
95    "humedad": 4095,
96    "id": 17
97  },
98  {
99    "temperatura": 277,
100   "humedad": 4095,
101   "id": 18
102  },
103  {
104    "temperatura": 288,
105    "humedad": 4095,
106    "id": 19
107  },
108  }
```

- Realizar el mismo procedimiento utilizando otra plataforma hardware.

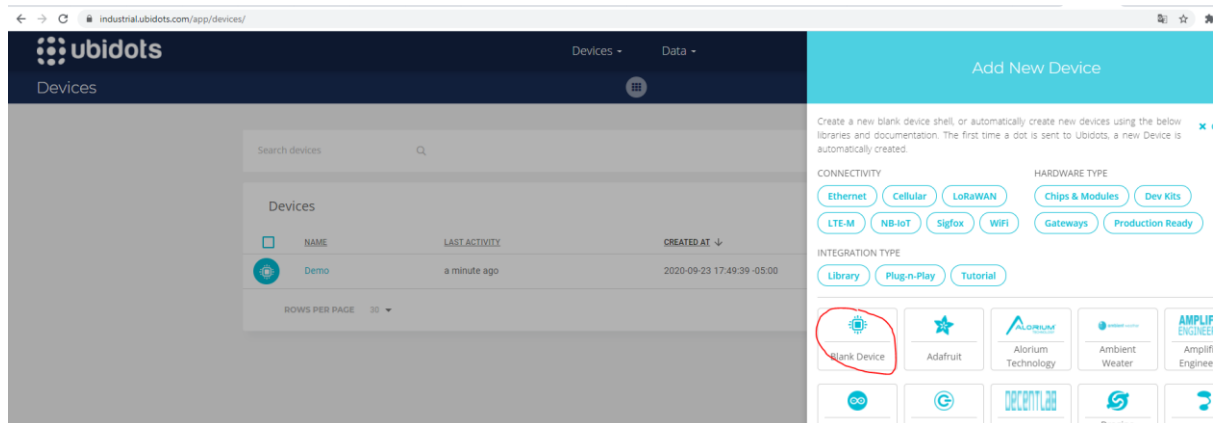
Procedimiento – Plataforma de IoT con soporte Rest

- Usar una plataforma de IoT, que tenga soporte a Rest y configurarla para conectar los clientes probando los métodos GET y POST. Aquí muestro como conectarse a Ubidots.

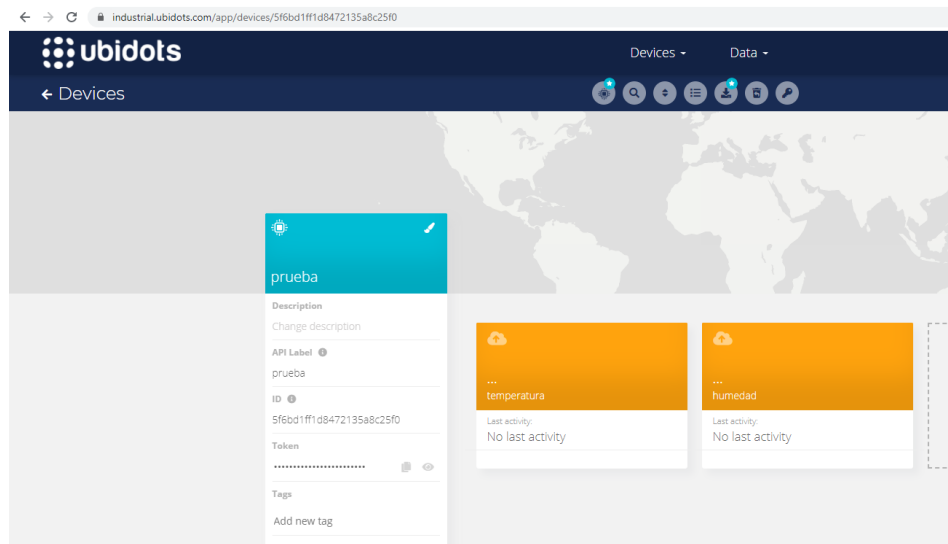
Lo primero es crear una cuenta en Ubidots.



Ya creada la cuenta, en la opción device del menú, añadimos un nuevo dispositivo, para iniciar nuestro proyecto. Seleccionaremos un blank device y le damos un nombre. En mi caso le he puesto prueba.



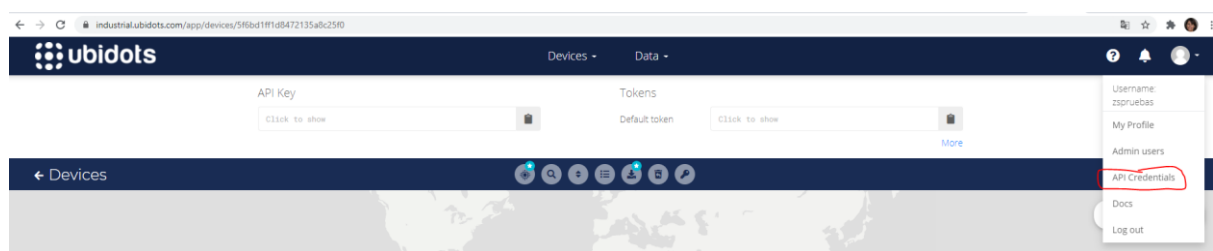
Al darle clic al dispositivo creado nos llevará a una nueva ventana en donde podemos crear variables, en este caso crearemos dos variables una para temperatura y otra para humedad.



Según la documentación encontrada en la página, la URL para acceder a través del método POST al servidor Rest está conformada de la siguiente manera:

<https://things.ubidots.com/api/v1.6/devices/nombredevice/?token=tutoken>

En nombredevice va el nombre del dispositivo y en tutoken va el token el cual se encuentra en la información de la cuenta del usuario.



Para recibir datos, método GET, se debe especificar en la URL de cual variable se requiere la información (nombrevariable), se pueden obtener todos los datos o solo el último valor. La URL quedaría de la siguiente manera:

Todos: <http://things.ubidots.com/api/v1.6/devices/nombredevice/nombrevariable/values?token=tutoken>

El último: <http://things.ubidots.com/api/v1.6/devices/nombredevice/nombrevariable/lv?token=tutoken>

Ahora nos conectamos a ubidots a través de Postman:

POST <http://things.ubidots.com/api/v1.6/devices/prueba/?token=BBFF-pxxcvXMqWOJLEPWfbiV8SxPNXZ9uqw> Send

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded **raw** binary GraphQL **JSON**

```

1 {
2   "temperatura":24,
3   "humedad":45
4 }

```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 1383 ms Size: 284 B Save

Pretty Raw Preview Visualize **JSON**

```

1 {
2   "humedad": [
3     {
4       "status_code": 201
5     }
6   ],
7   "temperatura": [
8     {
9       "status_code": 201
10    }
11  ]
12 }

```

Los datos se verán reflejados en el servidor:

[←](#) [→](#) [↻](#) industrial.ubidots.com/app/devices/5f6bd1ff1d8472135a8c25f0

ubidots Devices Data

[←](#) Devices

prueba

Description
Change description

API Label 🔍
prueba

ID 🔍
5f6bd1ff1d8472135a8c25f0

Token
..... 🔍

Tags
Add new tag

Last activity

45.00
humedad
Last activity: a minute ago

24.00
temperatura
Last activity: a minute ago

VARIABLES PER PAGE 10 🔍

Probemos el GET:

The first screenshot shows a GET request to `http://things.ubidots.com/api/v1.6/devices/prueba/temperatura/values/?token=BBFF-pxxcvXMqWQJLEPWft...` with a status of 200 OK. The response is a JSON object:

```
{
  "count": true,
  "next": null,
  "previous": null,
  "results": [
    {
      "timestamp": 1600902613008,
      "value": 24.0,
      "context": {},
      "created_at": 1600902613008
    }
  ]
}
```

The second screenshot shows a GET request to `http://things.ubidots.com/api/v1.6/devices/prueba/temperatura/lv/?token=BBFF-pxxcvXMqWQJLEPWfbiV8S...` with a status of 200 OK. The response is a plain text value:

```
24.0
```

6. En la plataforma hardware (ESP32) el sketch quedaría de la siguiente manera, cambiando únicamente la URL a la cual se debe conectar.

```
#include <HTTPClient.h>
#include <WiFi.h>
#include <ArduinoJson.h>

const char* ssid = "*****"; //El SSID de la red wifi a la que se conectará
const char* password = "*****"; //El password para conectarse a la red
inalambrica

void setup() {
  Serial.begin(115200); //Serial connection
  setup_wifi(); //WiFi connection
  delay(1500);
}

void setup_wifi() {

  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
```

```

WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}

Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
}

void loop() {

    int temp = analogRead(34); //lectura de los sensores en los pines 34 y 35
    int hum = analogRead(35);

    String variable;

    DynamicJsonDocument doc(1024); //creacion del json

    doc["temperatura"] = temp;
    doc["humedad"] = hum;

    serializeJson(doc, variable);
    Serial.println("dato a enviar: "+ variable);

    HTTPClient http;    //Declare object of class HTTPClient
    WiFiClient client;
    //Specify request destination
    http.begin(client,
"http://things.ubidots.com/api/v1.6/devices/prueba/?token=BBFF-
pxxcvXMqW0JLEPWfbiV8SxPNXZ9uqw");
    http.addHeader("Content-Type", "application/json"); //Specify content-
type header

    int httpCode = http.POST(variable);    //Send the request
    String payload = http.getString();    //Get the response payload

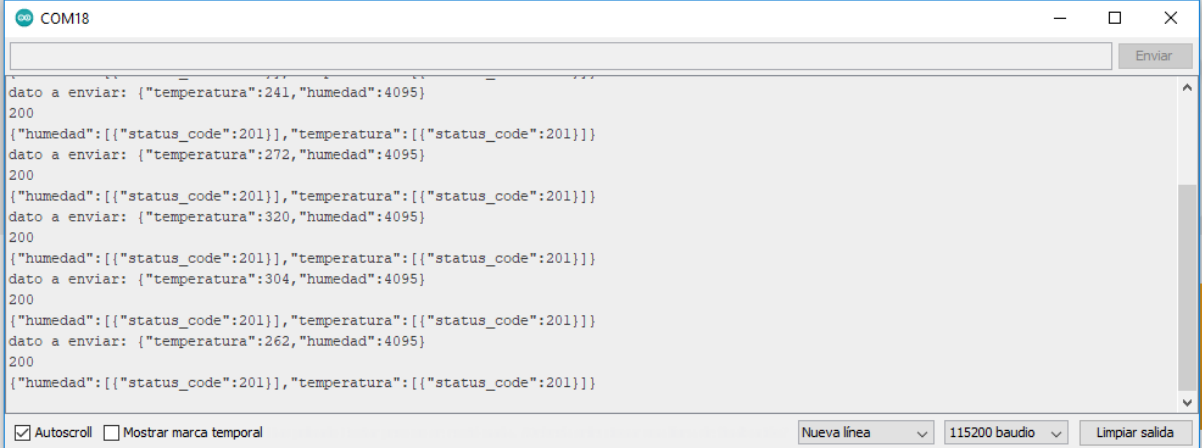
    Serial.println(httpCode);    //Print HTTP return code
    Serial.println(payload);    //Print request response payload

    http.end();    //Close connection

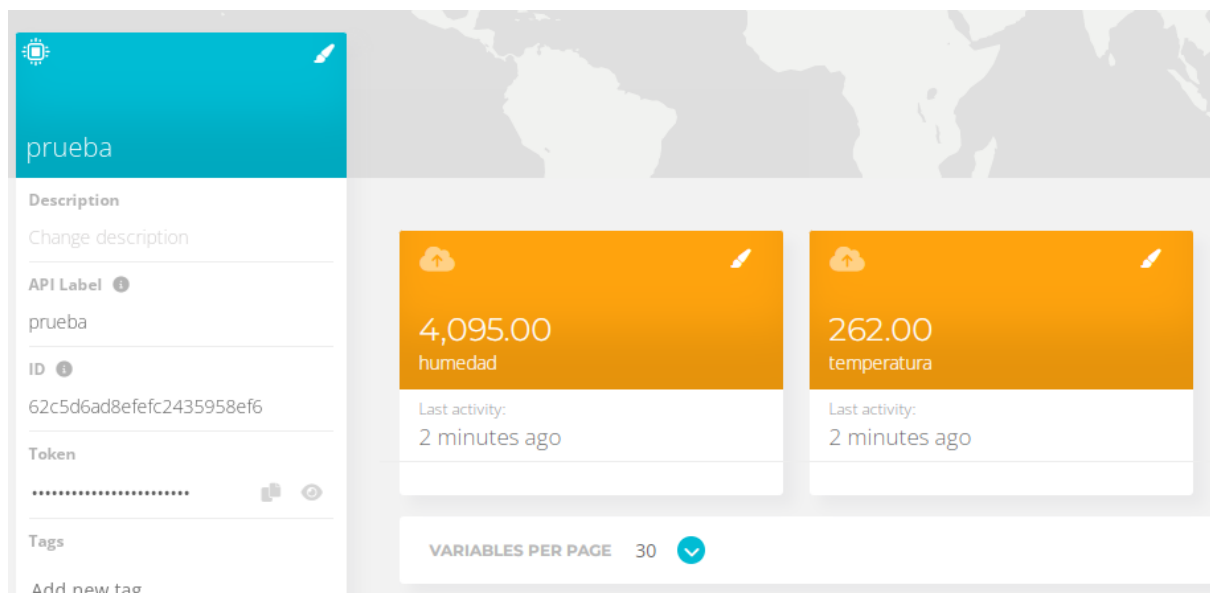
    delay(5000);    //Send a request every 5 seconds
}

```

Se observan los datos enviados por el ESP32 y recibidos en Ubidots



```
dato a enviar: {"temperatura":241,"humedad":4095}
200
{"humedad":{"status_code":201},"temperatura":{"status_code":201}}
dato a enviar: {"temperatura":272,"humedad":4095}
200
{"humedad":{"status_code":201},"temperatura":{"status_code":201}}
dato a enviar: {"temperatura":320,"humedad":4095}
200
{"humedad":{"status_code":201},"temperatura":{"status_code":201}}
dato a enviar: {"temperatura":304,"humedad":4095}
200
{"humedad":{"status_code":201},"temperatura":{"status_code":201}}
dato a enviar: {"temperatura":262,"humedad":4095}
200
{"humedad":{"status_code":201},"temperatura":{"status_code":201}}
```



7. Seleccione otra plataforma hardware y realice el sketch para conectarse a Ubidots y publicar el json con los datos de los sensores, de la misma manera como se hizo con el ESP32.

ENTREGA

Realiza un documento en donde se evidencie todo el procedimiento realizado.