TALLER 6. Servidor IoT - Local

Inicio

El objetivo de este taller es entender los conceptos relacionados con el desarrollo de un servidor de IoT, que implemente mecanismos para recibir datos usando los protocolo MQTT y REST. Se trabajará en un lenguaje de programación que facilite el desarrollo. En este caso trabajaremos con java script.

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

El desarrollo del taller se realizará en 3 fases:

1. Implementación del servidor con soporte a MQTT
2. Implementación del servidor con soporte a REST
3. Integración con el objeto IoT desarrollado

Creación del servidor IoT Local

1. Instalación de Node.js

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

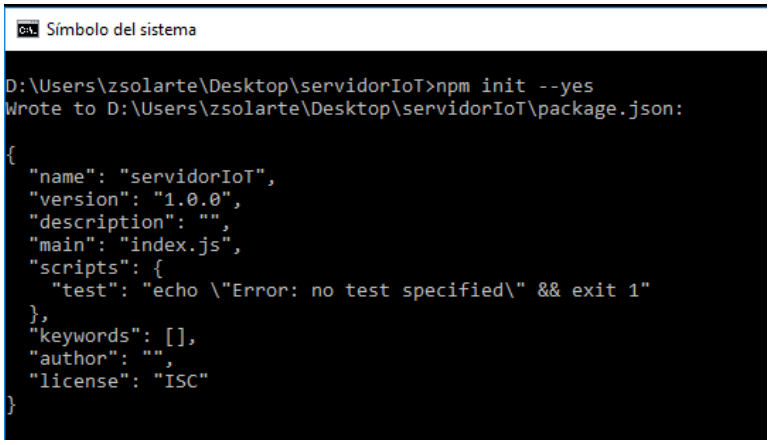
Una vez descargado el archivo procedemos con la instalación, que no es más que seguir el wizard y dar Next. Junto con Node.js se instalará el gestor de paquetes NPM, que es una herramienta que facilita la instalación de paquetes necesarios para nuestros proyectos.

2. Creación y configuración del proyecto.

Creamos una carpeta para almacenar nuestro proyecto, en este caso la he llamado **servidorIoT** y la he creado en el escritorio. Abrimos el cmd y nos ubicamos en la carpeta que acabamos de crear, y ejecutamos el comando:

npm init –yes

Este comando lo que hace es crear un json (package.json) con toda la descripción de proyecto y además muestra los paquetes instalados.

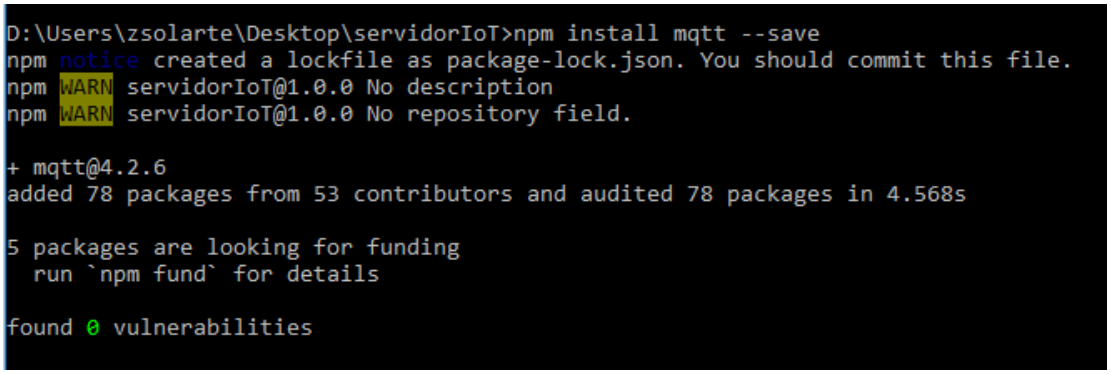


Servidor IoT con soporte a MQTT

3. Instalamos de las librerías requeridas.

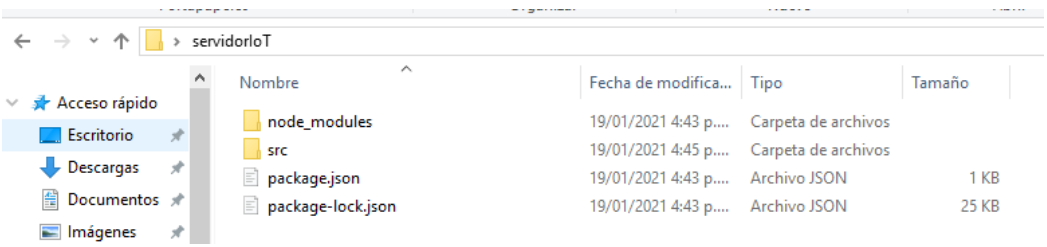
Si instala la librería de MQTT que nos permitirá conectarnos a un bróker MQTT y realizar publicaciones y subscripciones. Para esto usamos el comando

npm install mqtt --save



Para tener más organizado el proyecto creamos dentro de la carpeta de nuestro proyecto una carpeta en donde pondremos todo el código que desarrollaremos. En este ejemplo la he llamado src.

Hasta este momento tendríamos lo siguiente:



4. Desarrollo del código para subscribirse y realizar publicaciones a un tópico dado.

Para este ejemplo usaremos mosquitto, el cual como ya lo habíamos trabajado se encuentra en la misma máquina (localhost).

Dentro de la carpeta src creamos un archivo con el nombre indexmqtt.js que será el que contenga el código principal de nuestros servicios mqtt.

El contenido de indexmqtt.js será el siguiente:

```
var mqtt = require('mqtt')
var client  = mqtt.connect('mqtt://localhost')

client.on('connect', function () {
  client.subscribe('topico1', function (err) {
    if (err) {
      console.log("error en la subscripcion")
    }
  })
})

client.on('message', function (topic, message) {
  // message is Buffer
  console.log(message.toString())
  client.publish('topico2', 'mensaje recibido')
  //client.end()  //si se habilita esta opción el servicio termina
})
```

5. Prueba del servicio desarrollado

Para hacer la prueba debemos tener activo mosquitto y usar algunos clientes, en este caso probaremos con la línea de comandos de mosquitto y mqtt lens.

Lo primero que debemos hacer es lanzar el servicio, esto lo hacemos con el siguiente comando:

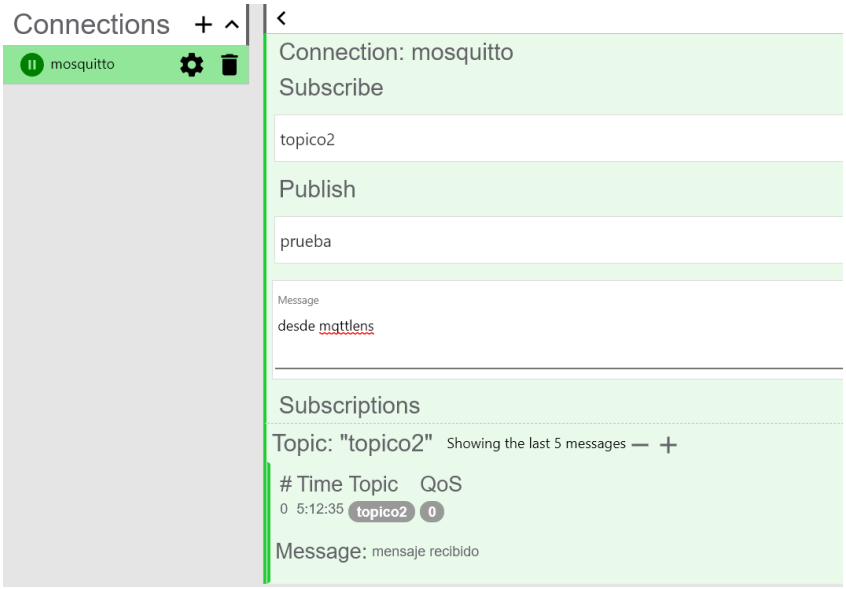
node src/indexmqtt.js

```
D:\Users\zsolarte\Desktop\servidorIoT>node src/indexmqtt.js
```

Él se quedará ahí esperando publicaciones:

Realizamos entonces una publicación desde cmd de mosquitto al tópico topico1 y configuramos mqttLens para que se subscriba al tópico topico2.

```
C:\Archivos de programa\mosquitto>mosquitto_pub -t topico1 -m "mensaje de prueba"
```



Se puede ver en el servidor que se imprime el mensaje recibido.

```
D:\Users\zsolarte\Desktop\servidorIoT>node src/indexmqtt.js
mensaje de prueba
```

6. Envío de datos al servidor desde el nodo desarrollado (hardware + sensores)

Utilizamos el mismo sketch usado en el taller pasado, ya que de igual manera estamos usando mosquitto como bróker, pero quien se suscribe es el servidor. Tener en cuenta el tópico que se está usando.

Para este ejemplo usaremos el ESP32 y el sketch será el siguiente:

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

#define mqttUser ""
#define mqttPass ""
#define mqttPort 1883

const char* ssid = "xxxxx";
const char* password = "xxxxxxxx";

char mqttBroker[] = "xxxxx"; //ip del servidor
char mqttClientId[] = "ArduinoYun01"; //cualquier nombre
char inTopic[] = "topico1";


void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
```

```
  for (int i=0;i<length;i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();
}

WiFiClient BClient;
PubSubClient client(BClient);

void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (client.connect(mqttClientId, mqttUser, mqttPass)) {
      Serial.println("connected");
      // Once connected, publish an announcement...
      int temp = analogRead(34);
      int hum = analogRead(35);
      String variable;

      StaticJsonDocument<256> doc;

      doc["temperatura"] = temp;
      doc["humedad"] = hum;
      doc["idnodo"] = 1;
      doc["timestamp"] =  1655133862;

      serializeJson(doc, variable);
      int lon = variable.length()+1;
      Serial.println(variable);
      char datojson[lon];
      variable.toCharArray(datojson, lon);
      client.publish(inTopic,datojson);
      client.disconnect();
      delay(5000);
      // ... and resubscribe
      //client.subscribe("topic2");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}

void setup()
```

```
{
  Serial.begin(115200); //Serial connection
  setup_wifi();   //WiFi connection
  client.setServer( mqttBroker, mqttPort );
  client.setCallback( callback );
  Serial.println("Setup done");
  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()
{
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
}
```

Se verifica el envío de los datos y que se reciben en el servidor:


```
PS D:\Users\zsolarte\Desktop\ServidorIoT> node src/indexmqtt.js
{"temperatura":293,"humedad":4095,"idnodo":1,"timestamp":1655133862}
{"temperatura":311,"humedad":4095,"idnodo":1,"timestamp":1655133862}
{"temperatura":297,"humedad":4095,"idnodo":1,"timestamp":1655133862}
{"temperatura":271,"humedad":4095,"idnodo":1,"timestamp":1655133862}
{"temperatura":261,"humedad":4095,"idnodo":1,"timestamp":1655133862}
```

Servidor IoT con soporte a REST

7. Instalación de las librerías requeridas.

Se deben instalar los paquetes express y morgan con el siguiente comando:

npm i express morgan

```
D:\Users\zsolarte\Desktop\servidorIoT>npm i express morgan
npm WARN servidorIoT@1.0.0 No description
npm WARN servidorIoT@1.0.0 No repository field.

+ express@4.17.1
+ morgan@1.10.0
added 64 packages from 37 contributors and audited 142 packages in 5.491s

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

found 0 vulnerabilities
```

Express facilita a creación del servicio rest desde node y morgan permite ver las peticiones y respuestas del servidor en la consola lo cual facilita el proceso de depuración, ya que nos permite ver los errores.

8. Desarrollo de las funciones GET y POST

Dentro de la carpeta src creamos un archivo con el nombre **indexrest.js** que será el que contenga el código principal de nuestros servicios.

Para que el proyecto quede mejor estructurado vamos a crear las rutas en un archivo aparte y desde nuestro archivo simplemente lo vamos a llamar. Para ello vamos a crear otra carpeta dentro de src y la vamos a llamar **rutas**, y dentro de ella vamos a crear otro archivo de nombre **ejemplo.js**

El contenido de indexrest.js será el siguiente:

```
const express = require('express'); //se indica que se requiere express
const app = express(); // se inicia express y se instancia en una constante de
nombre app.
const morgan = require('morgan'); //se indica que se requiere morgan

// settings
app.set('port', 3000); //se define el puerto en el cual va a funcionar el
servidro

// Utilities
app.use(morgan('dev')); //se indica que se va a usar morgan en modo dev
app.use(express.json()); //se indica que se va a usar la funcionalidad para
manejo de json de express

//Routes
app.use(require('./rutas/ejemplo.js'));

//Start server
app.listen(app.get('port'), ()=> {
    console.log("Servidor funcionando");
}); //se inicia el servidor en el puerto definido y se pone un mensaje en la
consola.
```

El contenido de ejemplo.js será el siguiente:

```
const { Router } = require('express');
const router = Router();

router.get('/', (req, res) => {
    res.json({ 'temp': 23, 'hum': 43 });
});

router.post('/', (req, res) => {
    console.log(req.body);
    res.send("datos recibidos....");
});

module.exports = router;
```

Para lanzar el servidor y ejecutar el código desarrollado se debe ejecutar el siguiente comando:

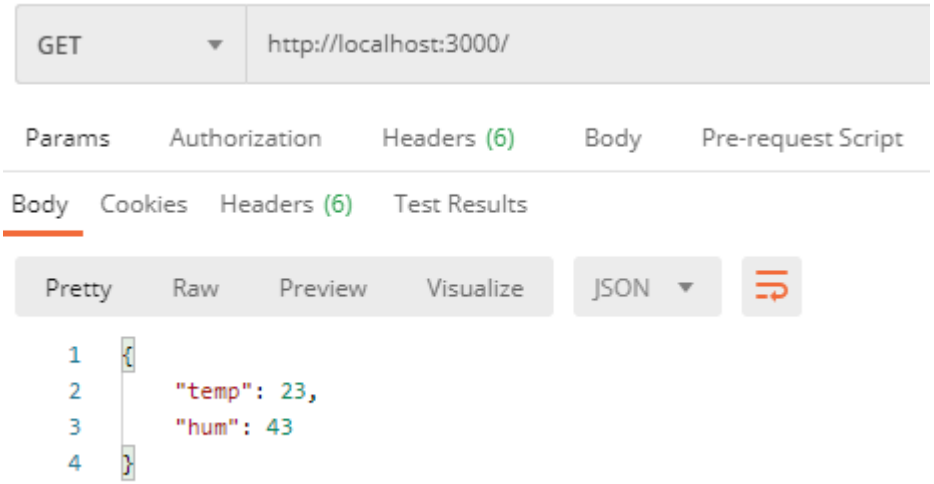
node src/indexrest.js

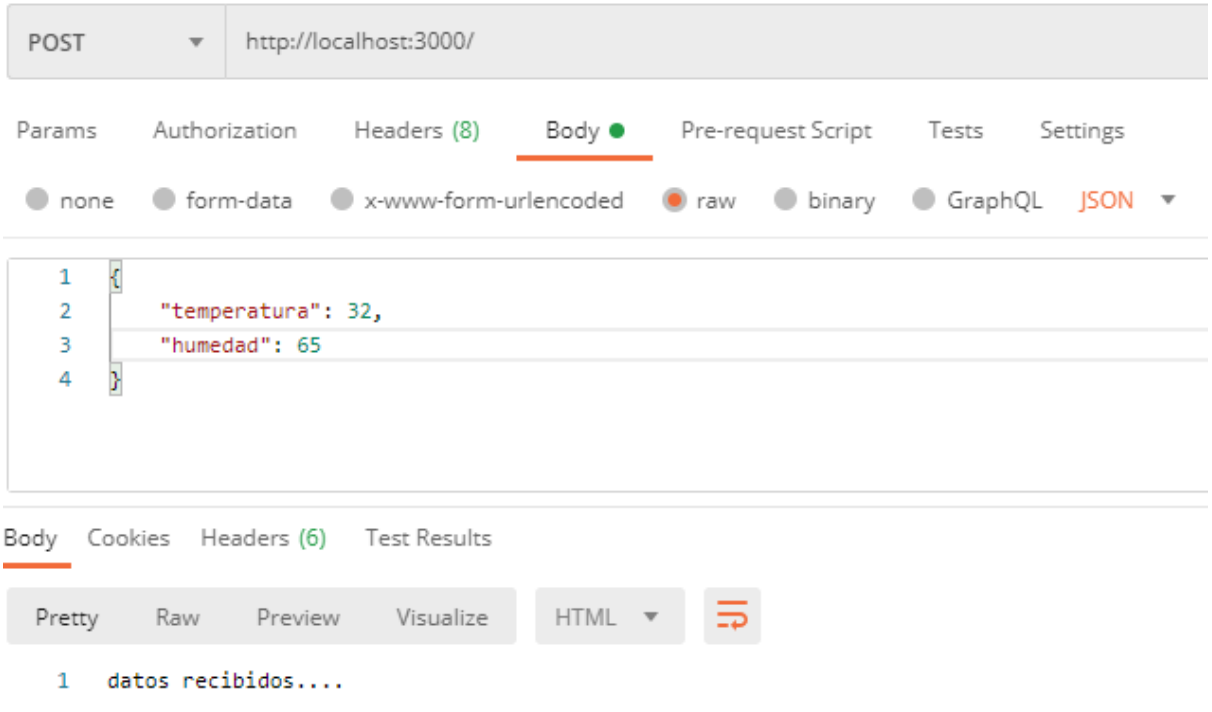
Se verá los siguiente:

```
D:\Users\zsolarte\Desktop\servidorIoT>node src/indexrest.js
Servidor funcionando
```

Desde postman se pueden probar los servicios:

La URL para hacerlo será: **http://localhost:3000**





En el servidor se verá lo siguiente:

```
D:\Users\zsolarte\Desktop\servidorIoT>node src/indexrest.js
Servidor funcionando
GET / 200 2.470 ms - 20
{ temperatura: 32, humedad: 65 }
POST / 200 12.160 ms - 19
```

9. Envío de datos al servidor desde el nodo desarrollado (hardware + sensores)

Utilizamos el mismo sketch usado en el taller pasado modificando la URL de conexión al servidor.

Para este ejemplo usaremos el ESP32 y el sketch será el siguiente:

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

const char* ssid = "xxxx";
const char* password = "xxxxxxx";

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);
  int hum = analogRead(35);
```

```
  String variable;

  DynamicJsonDocument doc(1024);

  doc["temperatura"] = temp;
  doc["humedad"] = hum;
  doc["idnodo"] = 1;
  doc["timestamp"] =  1655133862;

  serializeJson(doc, variable);

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

  HTTPClient http;
  WiFiClient client;
  http.begin(client, "URL del servidor");
  http.addHeader("Content-Type", "application/json", 0, 0);

  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 30 seconds
}
```

Se verifica que los datos se envían y reciben en el servidor.

```
PS D:\Users\zsolarte\Desktop\ServidorIoT> node src/indexrest.js
Servidor funcionando
{ temperatura: 295, humedad: 4095, idnodo: 1, timestamp: 1655133862 }
POST / 200 209.395 ms - 19
{ temperatura: 299, humedad: 4095, idnodo: 1, timestamp: 1655133862 }
POST / 200 47.718 ms - 19
{ temperatura: 302, humedad: 4095, idnodo: 1, timestamp: 1655133862 }
POST / 200 9.809 ms - 19
{ temperatura: 289, humedad: 4095, idnodo: 1, timestamp: 1655133862 }
POST / 200 48.610 ms - 19
{ temperatura: 272, humedad: 4095, idnodo: 1, timestamp: 1655133862 }
POST / 200 47.148 ms - 19
{ temperatura: 259, humedad: 4095, idnodo: 1, timestamp: 1655133862 }
POST / 200 52.204 ms - 19
```

ENTREGA

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