



SEP

SECRETARÍA DE
EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO
Instituto Tecnológico de San Juan del Río

Instituto Tecnológico de San Juan del Río



Web Api concurrente con dispositivo de IoT

P R E S E N T A:

**Luna González Rocio
Mendoza Trejo Jairo
Robles Padilla Oswaldo
Valencia Valencia Mauricio**

Ingeniería en sistemas computacionales

Almacenamiento en paralelo para IOT

San Juan del Río, Qro. Noviembre de 2020



Av. Tecnológico No. 2, Col. Centro, C.P. 76800
San Juan del Río, Qro. Conmutador: (01) 427 27 2 85 46, 427 27 2 41 18 Ext. 123
e-mail: divisionestudiosp@yahoo.com.mx
www.itsanjuan.edu.mx



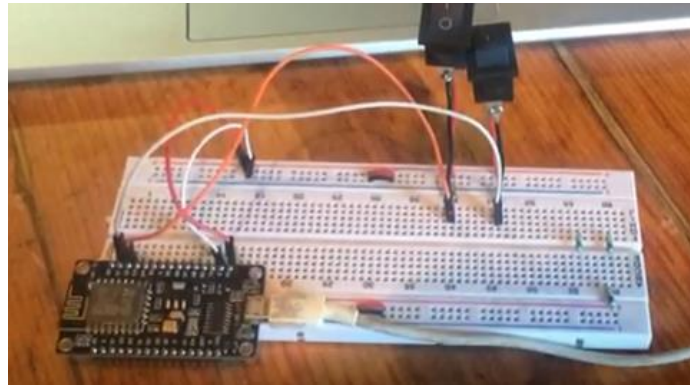
Web API Concurrente con Dispositivo IoT.

1. Objetivo.

Desarrollar una prueba de concepto de un servicio web que recibe información en paralelo desde diferentes dispositivos IoT.

2. Materiales.

- nodemcu
- Interruptores ON/OFF
- Resistencias
- Laptop
- Cables para protoboard



3. Software.

Para realizar el mini proyecto se utilizaron los siguientes:

MongoDB

Esta base de datos NoSQL ofrece una gran escalabilidad y flexibilidad, MongoDB almacena datos en documentos flexibles similares a JSON, por lo que los campos pueden variar entre documentos y la estructura de datos puede cambiarse con el tiempo.



Arduino IDE

El software Arduino (IDE) de código abierto facilita la escritura de código y su carga en la placa, NodeMCU ESP8266. Funciona en Windows, Mac OS X y Linux. El entorno está escrito en Java y basado en Processing y otro software de código abierto



NodeJS

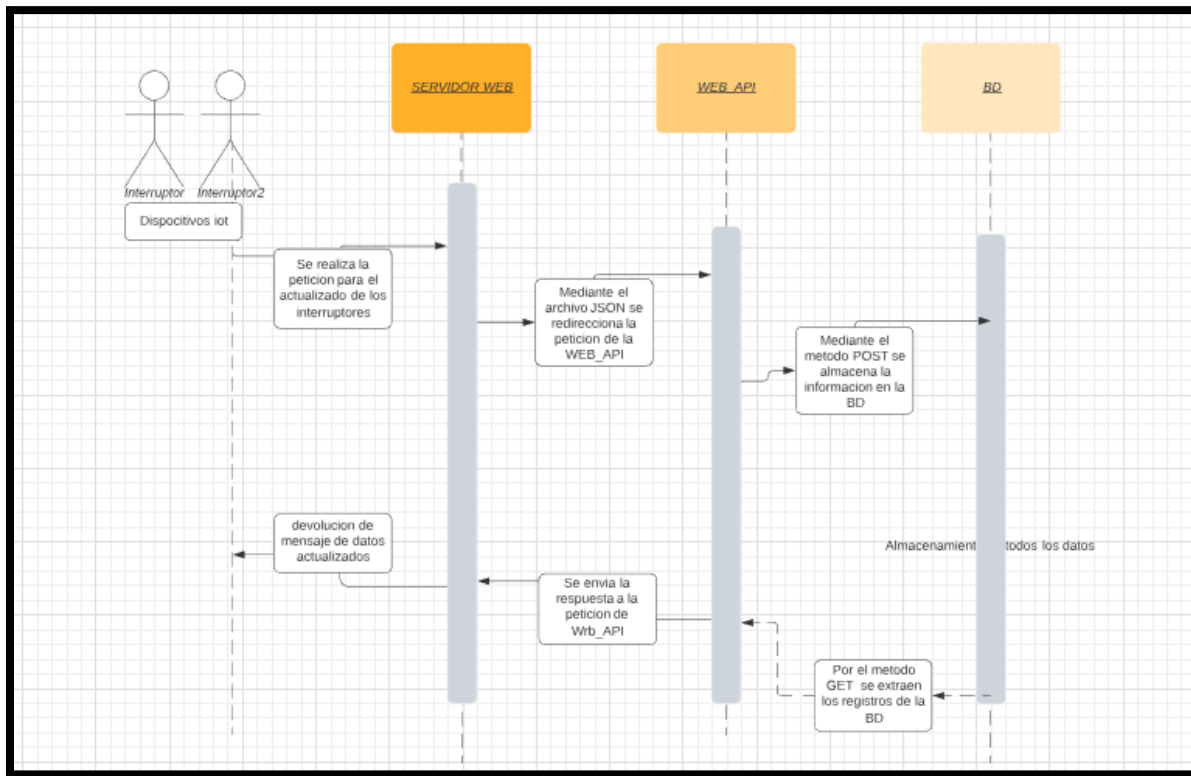
Es un entorno JavaScript de lado de servidor que utiliza un modelo asíncrono y dirigido por eventos. Node.js está diseñado para crear aplicaciones network escalables, pueden atenderse muchas conexiones simultáneamente. Por cada conexión, se activa la devolución de llamada o *callback*, pero si no hay trabajo que hacer, Node.js se “dormirá”. Esto contrasta con el modelo de concurrencia más común de hoy en día, en el que se emplean hilos del Sistema Operativo.

4. Diagrama UML.

En nuestro dispositivo iot está conformado por dos interruptores on/off. Los cuales cada uno hace su cambio de estado correspondiente se envía la petición al servidor web para actualizar el estado de los mismos, tenemos el servidor web este envía desde el dispositivo iot hacia el web api la cual contiene NodeJS en un archivo tipo json el cual contiene la información del estado de los dos interruptores estando conectados por medio de la red hacia nuestro servidor de web_api.

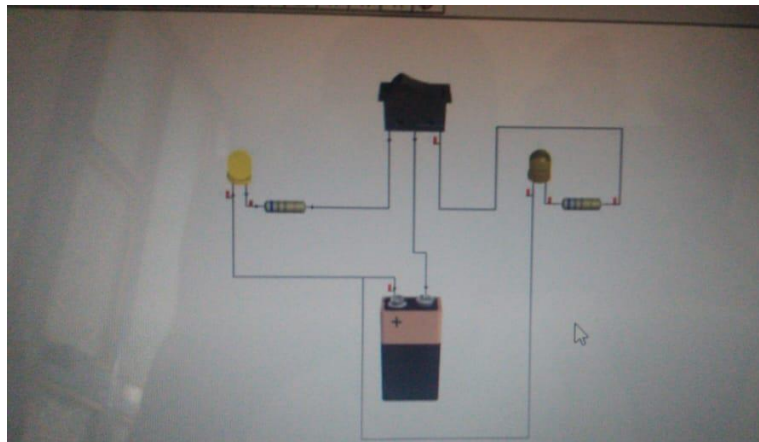
En nuestro servidor de base de datos MongoDB se estará recibiendo las peticiones de nuestro dispositivo iot con web_api esto lo hace mediante los métodos de POST y GET el método POST se utilizó para el almacenamiento de la información en la base de datos y el método GET para la extracción de todos los registros de la base de datos.



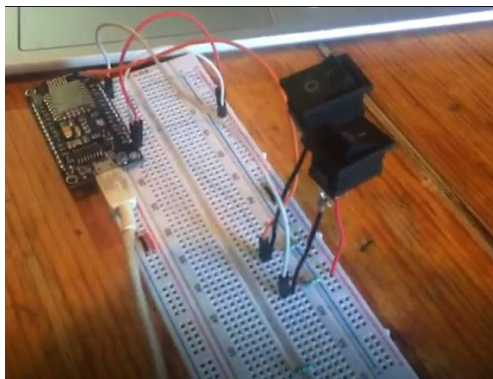


5. Armado del circuito

Los interruptores de 3 estados que se utilizaron se caracterizan por tener 3 entradas como su nombre lo dice los cuales son corriente, señal y tierra, en este caso la terminal de la corriente se encargará de mandar la información sobre el estado del interruptor.



Montamos el NodeMCU sobre la protoboard, y para dar alimentación a los interruptores en y cargar el programa al nodemcu para probar su funcionamiento



6. Código

Una vez montado el circuito para el código se utilizaron dos métodos get y post para recibir los valores de los pulsadores si estaban encendido o apagado con la placa nodemcu cuenta con una conexión a wifi la información se envía a la base de datos que se encuentra en mongo db y de esta forma se puede verificar que se está guardando la información correctamente se incluyeron las librerías de arduino json y la de wifi para poder conectarse se proporcionaron las credenciales de acceso y se definieron los estados de los apagadores los cuales cuando sufren un cambio de encendido y apagado se compara en los ciclos y si este es verdadero con la acción en el ciclo este nos envía el estado del apagador mediante el metodo get y post

api

```
#include <ArduinoJson.h>
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <WiFiClient.h>

String server= "";

const char *ssid = "INFINITUMF3RV_2.4";
const char *password = "vfA2QQq39N";
//const char *ssid = "UbeeA3E4-2.4G";
//const char *password = "40BBFCA3E4";

//-----
#define BTN1 5//D2
#define BTN2 4//D3

int estado=0;
int estado2=0;
int estado3=0;
int estado4=0;
```



```

void setup()
{

    Serial.begin(9600);

    pinMode(BTN1, INPUT);
    pinMode(BTN2, INPUT);

    Serial.println("WiFi connected");
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);

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

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

void loop()
{
    if (WiFi.status() == WL_CONNECTED)
    {
        if (digitalRead(BTN1) == HIGH) {

```



```

        if (estado == 0) {
            Serial.println("BT1-ENCENDIDO");
            post("BT1-ENCENDIDO");
            estado=1;
            estado2=0;
        }
    }
    else if (digitalRead(BTN1) == LOW){

        if (estado2 == 0) {
            Serial.println("BT1-APAGADO");
            post("BT1-APAGADO");
            estado=0;
            estado2=1;
        }
    }
    if (digitalRead(BTN2) == HIGH) {

        if (estado3 == 0) {
            Serial.println("BT2-ENCENDIDO");
            post("BT2-ENCENDIDO");
            estado3=1;
            estado4=0;
        }

    }
    else if (digitalRead(BTN2) == LOW){

```



```

        if (estado4 == 0) {
            Serial.println("BT2-APAGADO");
            post("BT2-APAGADO");
            estado3=0;
            estado4=1;
        }

    }
    delay(1000);
}

}

void post(String entrada) {
    Serial.println("Inicio post");
    HTTPClient http;
    String json;
    server ="http://192.168.1.73:3000/api/product";

    StaticJsonDocument<256>doc;
    doc["entrada"]= String(entrada);
    serializeJson(doc, json);

    http.begin(server);
    http.addHeader("Content-Type", "application/json");
    http.POST(json);
    http.writeToStream(&Serial);
    http.end();

    Serial.println ("Termino post");
}

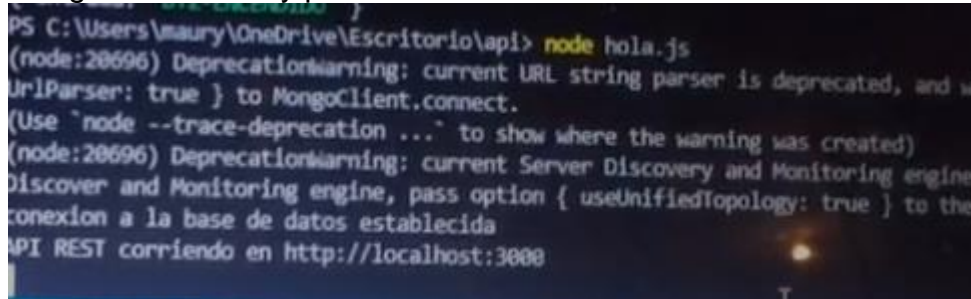
```



7. Pruebas

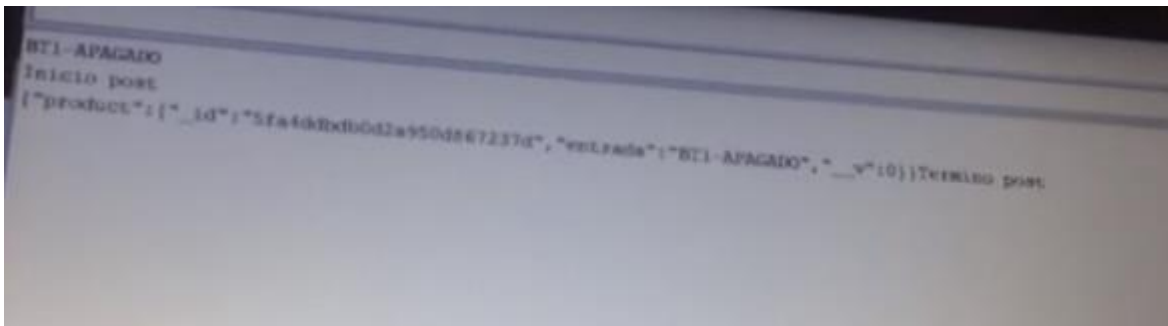
Para verificar el correcto funcionamiento del dispositivo IoT y la web api se realizo la siguiente prueba

Se cargó el programa de api al nodemcu ya montado en nuestro circuito
En el api rest cargar el servidor y ponerlo en funcionamiento



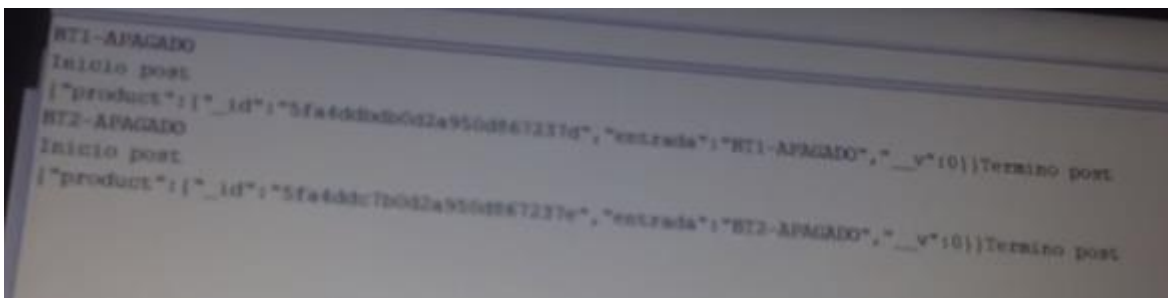
```
PS C:\Users\maury\OneDrive\Escritorio\api> node hola.js
(node:28696) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version. To avoid deprecation warnings, use the new URLParser: true } to MongoClient.connect.
(Use 'node --trace-deprecation ...' to show where the warning was created)
(node:28696) DeprecationWarning: current Server Discovery and Monitoring engine is deprecated, and will be removed in a future version. To avoid deprecation warnings, use the new Server Discovery and Monitoring engine, pass option { useUnifiedTopology: true } to the MongoClient constructor.
conexion a la base de datos establecida
API REST corriendo en http://localhost:3000
```

Una vez cargado el servidor y puesto en funcionamiento aparecerá la información del puerto al que nos conectamos en este caso el 3000 y que la conexión fue establecida
Como el api ya está cargada empezaremos a apagar los interruptores para comprobar que funciona correctamente



```
{
  "product": {
    "_id": "5fa4db8b0d2a950d867237d",
    "entrada": "BT1-APAGADO",
    "v": 10
  }
}
```

Cuando presionemos el interruptor nos mandara el mensaje donde nos indicara el estado del mismo en este caso fue el botón uno, pero de igual forma lo haremos con el botón dos



```
{
  "product": {
    "_id": "5fa4db8b0d2a950d867237e",
    "entrada": "BT2-APAGADO",
    "v": 10
  }
}
```

Los resultados se envían al servidor y este a la base de datos para poder consultarlos en el servidor aparecen de la siguiente manera

```

API REST corriendo en http://localhost:3000
POST /api/product
[Object: null prototype] { entrada: 'hola' }
POST /api/product
{ entrada: 'BT1-APAGADO' }
POST /api/product
{ entrada: 'BT2-APAGADO' }

```

Y en la base de datos nos marcara las últimas dos entradas solo debemos de actualizar para comprobar que los datos se mandaron de forma correcta

}

6	5fa452b2d6d22x915m0067237f	"BT1-APAGADO"
7	5fa452b2d6d22x915m0067237f	"BT2-APAGADO"

ADD DATA			
products			
	_id ObjectId	entrada string	_v Int32
1	5fa452b2d6d22x915m0067237f	"BT1-APAGADO"	0
2	5fa452b2d6d22x915m0067237f	"BT2-APAGADO"	0
3	5fa452b2d6d22x915m0067237f	"BT1-ENCENDIDO"	0
4	5fa452b2d6d22x915m0067237f	"BT2-ENCENDIDO"	0
5	5fa452b2d6d22x915m0067237f	"hola"	0
6	5fa452b2d6d22x915m0067237f	"BT1-APAGADO"	0
7	5fa452b2d6d22x915m0067237f	"BT2-APAGADO"	0

Concluyendo con las pruebas nos dimos cuenta que funcionan de manera correcta ya que cuando pulsamos los interruptores funcionan de manera correcta y esto se envía de manera correcta al servidor y a la base de datos de manera correcta comprobando así el funcionamiento del api.

