

Una solución IoT basada en una ESP-32, Servidor Mosquito, una base de datos SQL y una interfaz de usuario basada en Node-RED

Tarea 1. Planteamiento del problema a resolver

Se requiere implementar una solución IoT de sensado de temperatura y humedad que permita registrar las mediciones en una base de datos. Entre las restricciones se tienen:

- La base de datos debe ser MySQL.
- Se trata de un rediseño que partirá del diseño que ya se tiene de la práctica 7.
- Los datos a registrar son temperatura, humedad, fecha y hora de la medición, tipo de tarjeta que reporta la medición, tópico.

Tarea 2. Estudios previos

De las prácticas anteriores ya se tiene conocimiento de todos los elementos que se usan en la solución, excepto de las bases de datos MySQL. Solo resta concentrar esfuerzos en conquistar MySQL lo suficiente para crear y una base de datos sencilla.

Algunos consideraciones:

- Podemos imaginar una base de datos como una tabla de Excel que tiene hojas. Pero también tiene diferencias con Excel como las siguientes: no hay un solo archivo para esas hojas, aquí las hojas se llaman tablas, las celdas no llevan fórmulas, se requiere un visualizador aparte para ver los datos como una tabla, se han creados muchos comandos de javascript que facilitan el acceso de los datos desde páginas (en esto es equivalente a lo que google apps script logra con google sheet). En realidad, Excell es más comparable con phpmyadmin, mientras que MySQL es más bien comparable con los datos brutos que aparecen en el excel.
- También es válido decir que phpmyadmin es a MySQL lo que Node-RED es a Mosquito. En otras palabras, que legalmente una base de datos se crea a punta de comandos, como lo que uno hace cuando programa Python, pero phpmyadmin hace que el proceso sea más parecido a Excel.
- Todo dato debe llevar un ID único, que lo diferencia de los demás.
- La llave primaria. Es un término usado en MySQL para dejar declarar aquella columna que se considera que identifica de manera única cada dato.
- Por más de un nombre: el de la base de datos y el de cada tabla
- En MySQL una columna no tiene simplemente un nombre, por ejemplo ID, sino otros parámetros como (En Excel también se configuran tipo de datos y otras cosas, pero esa información no aparecen de manera explícita en la tabla):
 - **Nombre:** Por ejemplo temperatura
 - **Tipo:** entero, flotante, char, etc

- **Índice:** se declara si es un dato especial, por ejemplo, PRIMARY significa que se trata de un dato de identificación, que define de manera única a cada fila
- **A_I:** se chequea si el dato debe incrementarse de manera automática. Por ejemplo, a cada dato que llega se le asigna un ID que se va incrementando en uno
- **Predeterminado:** es cuando queremos indicar que la celda sea llenada automáticamente, por ejemplo, la fecha y hora es un dato que el sistema puede deducir y llenar

Paso 1. Conocimientos de la interfaz phpmyadmin

- **Prerrequisitos:**
 - Debe estar iniciada la VM donde está instalado MySQL y phpmyadmin
 - Debes conocer la IP pública o DNS de esa VM.
 - Debes conocer usuario y clave de MySQL. En nuestro caso, el usuario es root y la contraseña la misma usada para iniciar la VM
- **El proceso para abrir el editor de base de datos:** abre phpmyadmin en un navegador así: IP/phpmyadmin, por ejemplo:
<http://ec2-35-153-57-109.compute-1.amazonaws.com/phpmyadmin/>
- **Las configuraciones más generales:**
 - Server connection collation. Dejamos el valor por defecto, pero ten presente de usar siempre lo mismo. Ten en cuenta que, así como los seres humanos tenemos diferentes lenguajes, las bases de datos también tienen diferentes formas de guardar datos. Por lo tanto, lo que selecciones aquí procura usarlo siempre al menos para esa base de datos.



Fig. 1

- **Menú horizontal:** Explora el Menú horizontal para que conozcas qué servicios brinda phpmyadmin para configurar o aprovechar las bases de datos en MySQL.

Tarea 3. Diseño general

El esquema de la solución a implementar es el de la figura siguiente:

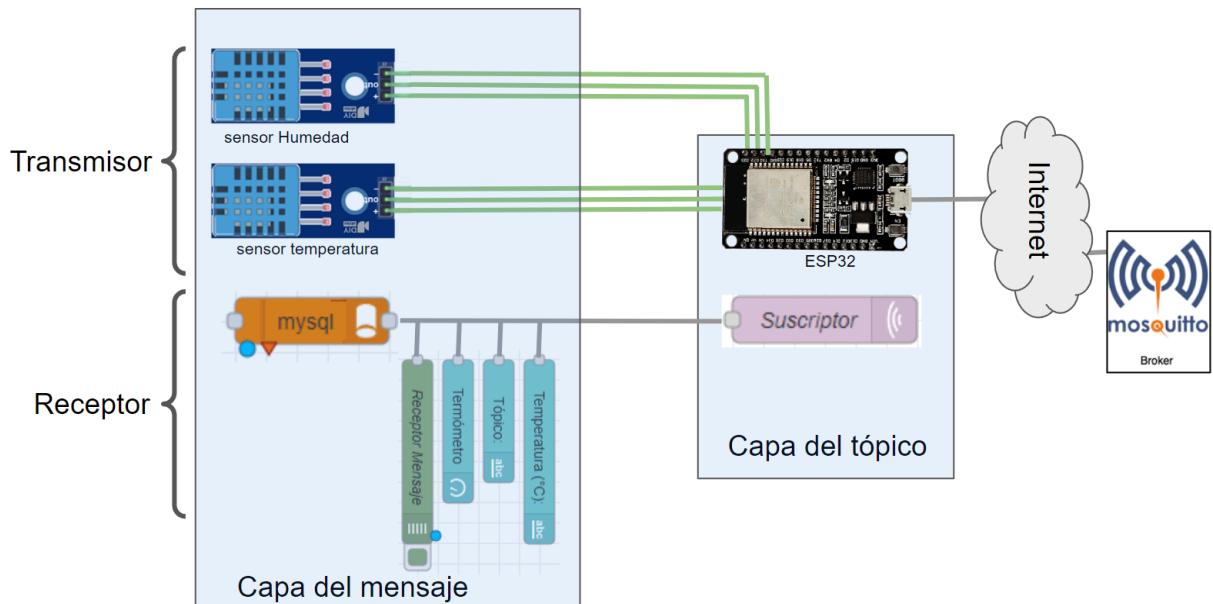
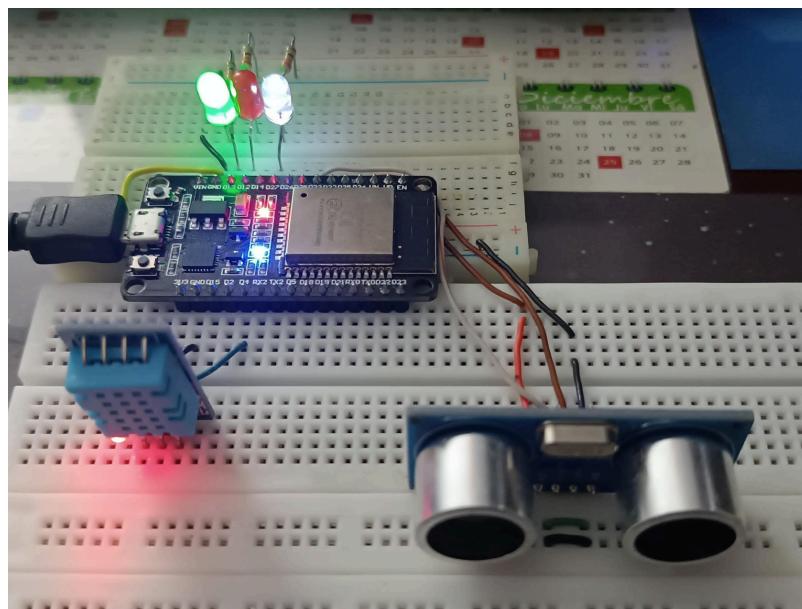


Fig. 2

De acuerdo a la figura de diseño, comparada con el montaje hecho en la práctica anterior, solo buscamos:

- El destino real del mensaje es la base de datos
- Los dispositivos (los que están verticales en el receptor) de visualización son solo eso: visualizadores

En mi caso utilizaré directamente un sensor **DHT11** y el sensor **HC-SR04**, para sensar temperatura, humedad y distancia respectivamente, el led interno de la ESP32 indicará la conexión al Wi-Fi, un led verde en el pin 13 indicará la correcta conexión a MQTT, en el pin 12 un led rojo que indicará cuando la conexión a MQTT falla, en el pin 27 un led blanco que se encenderá cada vez que el sensor HC-SR04 detecte una distancia menor a 20 cm y este está conectado a los pines 32 y 33, finalmente el sensor DHT11 estará conectado al pin 4 de la ESP32:



Tarea 4. Diseño de la base de datos

Paso 1. Diseño e implementación de base de datos inicial.

- Realizamos un diseño previo de la tabla que deseamos como base de datos inicial. El siguiente puede ser ese diseño:

Nombre de la DB		DB_Grupo_B_IOT			
Nombre de la tabla		mediciones2024			
nombre:ID Tipo: INT Indice: PRIMARY A_I: chequeado	nombre:Fecha Tipo: DATETIME Predeterminado : CURRENT_TIMESTAMP	nombre:Dispositivo Tipo: VARCHAR Longitud/Valores: 50	nombre: Temperatura Tipo: float	nombre: Humedad Tipo:float	nombre: Distancia Tipo:float

Para esta tarea, simplemente agregué una columna (la que está resaltada de amarillo) para los datos del sensor de distancia.

Tarea 5. Implementación de la base de datos

Paso 1. Creación y configuración de los campos

- Creamos la base de datos con nombre DB_Grupo_B_IOT



Bases de datos

Crear base de datos

utf8mb4_0900_ai_ci

Crear

- Creamos la tabla: mediciones2024

The screenshot shows the 'Crear tabla' (Create Table) dialog in MySQL Workbench. The 'Nombre:' field is set to 'mediciones2023' and the 'Número de columnas:' field is set to '5'. The 'Continuar' (Continue) button is visible at the bottom of the dialog.

- Configurar los encabezados de cada columna de la tabla. Lo único extraño aquí es que aparece una fila por cada columna a configurar

- Columna ID**

Nombre	Tipo	Índice	A.I
ID	INT	PRIMARY	<input checked="" type="checkbox"/>

Seleccionar desde las columnas centrales

- Columna Fecha**

Nombre	Tipo	Longitud/Valores	Predeterminado
Fecha	DATETIME		CURRENT_TIME

Seleccionar desde las columnas centrales

- Dispositivos**

Nombre	Tipo	Longitud/Valores
Dispositivos	VARCHAR	50

Seleccionar desde las columnas centrales

- Variables de Temperatura y Humedad**

Nombre	Tipo
Temperatura	FLOAT

Seleccionar desde las columnas centrales

Nombre	Tipo
Humedad	FLOAT

Seleccionar desde las columnas centrales

- NO OLVIDE:** Guardar

- Como resultado tenemos la estructura de la tabla**

		Estructura de tabla	Vista de relaciones						
#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
1	ID	int			No	Ninguna		AUTO_INCREMENT	Cambiar
2	Fecha	datetime			No	CURRENT_TIMESTAMP		DEFAULT_GENERATED	Cambiar
3	Dispositivos	varchar(50)	utf8mb4_0900_ai_ci		No	Ninguna			Cambiar
4	Temperatura	float			No	Ninguna			Cambiar
5	Humedad	float			No	Ninguna			Cambiar

- Paso 2.** El dia a dia

Con lo anterior, la base de datos ya está implementada, pero el problema no está resuelto porque resta lograr que a ella lleguen los datos de manera automática, es decir, falta conectarla al sistema IoT

En cuanto al día a día es simple: ella arranca cada vez que se reinicia la VM.

Se procede a crear la tabla que diseñó en la **Tarea 4**:

The image consists of three vertically stacked screenshots of the phpMyAdmin interface, all from the same session on a local host at port 3306.

- Screenshot 1:** Shows the "Bases de datos" (Databases) page. A modal window titled "Crear base de datos" is open, showing the database name "DB_Grupo_B_IOT" and character set "utf8mb4_0900_ai_ci". A "Crear" (Create) button is visible.
- Screenshot 2:** Shows the "Base de datos: DB_Grupo_B_IOT" page. It displays a message: "No se han encontrado tablas en la base de datos." (No tables have been found in the database). Below this, a "Crear nueva tabla" (Create new table) form is shown, with the table name "mediciones2024" and 6 columns selected. A "Crear" (Create) button is present.
- Screenshot 3:** Shows the "Estructura" (Structure) page for the "mediciones2024" table. The table has 6 columns defined:
 - ID:** INT, Primary Key (PRIMARY).
 - Fecha:** DATETIME.
 - Dispositivos:** VARCHAR(50).
 - Temperatura:** FLOAT.
 - Humedad:** FLOAT.
 - Distancia:** INT.

Finalmente se tiene creada la tabla **mediciones2024 en la base de datos **DB_Grupo_B_IOT**:**

The screenshot shows the 'Estructura de tabla' (Table Structure) page in phpMyAdmin. The table 'dagd_v2' is selected. The columns are listed as follows:

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
1	ID	int			No	Ninguna		AUTO_INCREMENT	Cambiar Eliminar Más
2	Fecha	datetime			No	CURRENT_TIMESTAMP		DEFAULT_GENERATED	Cambiar Eliminar Más
3	Dispositivos	varchar(50)	utf8mb4_0900_ai_ci		No	Ninguna			Cambiar Eliminar Más
4	Temperatura	float			No	Ninguna			Cambiar Eliminar Más
5	Humedad	float			No	Ninguna			Cambiar Eliminar Más
6	Distancia	int			No	Ninguna			Cambiar Eliminar Más

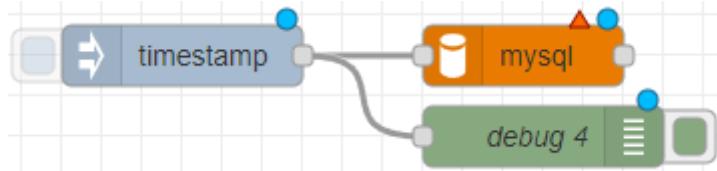
At the bottom, there are buttons for 'Seleccionar todo', 'Para los elementos que están marcados:', and other actions like 'Examinar', 'Cambiar', 'Eliminar', 'Primaria', 'Único', 'Índice', 'Espacial', 'Texto completo', and 'Agregar a columnas centrales'.

Tarea 6. Diseño de la conexión entre Mosquito y MySQL apoyados en Node-RED y phpmyadmin

Paso 1. Montaje para pruebas de conexión entre Mosquito y MySQL

El problema a resolver es: cómo logramos, sin recurrir al esquema final, sino de la forma más simple posible comprobar que sí es posible la conexión entre bloques IoT y la base de datos, pero sobre todo es debemos identificar con qué configuración se logra esto.

Paso 2. Creamos el flujo de trabajo IoT más simple posible, que sería el siguiente:



La idea es que:

- Al oprimir el botón izquierdo del bloque Inject (el que tiene el nombre timestamp), podemos ver, mediante phpmyadmin que los datos que entrega Inject se hayan escrito de la manera deseada en la base de datos.
- Mediante debug, podemos conocer cómo es el JSON que espera el bloque MySQL

Configuraciones:

- Bloque MySQL:

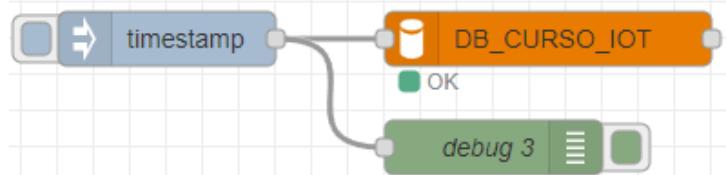
The screenshot shows the 'Properties' dialog for a MySQL connection in Node-RED. The fields are as follows:

Host	127.0.0.1
Port	3306
User	root
Password
Database	BD_CURSO_IOT
Timezone	+hh:mm
Charset	UTF8
Name	Name

- **Host:** es donde está ubicada la base de datos. En este caso es la misma VM que Node-RED y por eso se usa la IP 127.0.0.1 que es la usada en los casos en que el sistema apuntado está en la misma máquina.
- **Port:** 3306 es el puerto por defecto de MySQL
- **User:** root es el usuario que venimos manejando en MySQL
- **Password:** el de MySQL que para nosotros, por estrategia hemos elegido que sea el mismo de la VM
- **Database:** MySQL puede tener varias bases de datos, indicamos el nombre de la nuestra
- Los demás parámetros los dejamos por defecto.
- **Bloque debug:** Lo configuramos para que muestre todo el objeto JSON



- Como resultado tenemos el flujo con los bloques configurados así



Paso 3. Configuración del Bloque Inject (el marcado como timestamp)

Aquí está la configuración clave porque:

- Es el bloque que arma los datos de manera tal que "DB_Grupo_B_IOT" los acepte y los pase a la tabla correspondiente en MySQL
- Para conocer cuál es la entrada que espera un bloque MySQL hacemos esto:

Seleccionamos el bloque DB_Grupo_B_IOT > . Entonces vemos que la entrada de ese bloque:

- Debe ser en formato JSON
- Entre esos datos se distinguen:
 - **msg.topic** must hold the query for the database, and the result is returned in **msg.payload**.
 - **msg.payload** can contain an array of values to bind to the topic.
- De lo anterior, lo más relevante es que señala que los datos que entran a la base de datos deben llevar un query (en español - sentencia SQL) y que ella debe ir en el **msg.topic**. Una query es una orden para la base de datos.
- El problema que surge ahora es: ¿cómo es esa query? ¿dónde va?
 - Va en el bloque inject, en el parámetro **msg.topic**, porque ese es el bloque encargado de preparar los datos para el bloque MySQL
 - Para conocer cuál es la sentencia, podemos aprovechar que cuando usamos phpmyadmin para meter datos a la base de datos, no solo lo hace sino que además muestra la query usada. Entonces hacemos lo siguiente:
 - Vamos a phpmyadmin > Bases de datos > seleccionamos nuestra base de datos DB_Grupo_B_IOT > seleccionamos nuestra tabla mediciones2024-2 > Insertar
 - Llenamos los datos, pero no tocamos ID ni Fechas porque son valores que el mismo sistema asignará

Servidor: localhost:3306 » Base de datos: DB_CURSO_IOT » Tabla: mediciones2023

Columna	Tipo	Función	Nulo	Valor
ID	int			
Fecha	datetime			CURRENT_TIMESTAMP
Dispositivos	varchar(50)			ESP32
Temperatura	float			20.5
Humedad	float			89.3

- Oprimimos “Continuar”, es entonces cuando aparece la ventana que informa del cumplimiento de la orden, pero también la query usada.

✓ 1 fila insertada.
La Id de la fila insertada es: 1

```
INSERT INTO `mediciones2023` (`ID`, `Fecha`, `Dispositivos`, `Temperatura`, `Humedad`  
CURRENT_TIMESTAMP, 'ESP32', '20.5', '89.3');
```

[Editar en línea] [Editar] [Crear código PHP]

Ejecutar la(s) consulta(s) SQL en la tabla DB_CURSO_IOT.mediciones2023: ⚙

```
1 INSERT INTO `mediciones2023` (`ID`, `Fecha`, `Dispositivos`, `Temperatura`,  
`Humedad`) VALUES (NULL, CURRENT_TIMESTAMP, 'ESP32', '20.5', '89.3');
```

- Pero es valioso seguir bajando hasta “Consola”, abrir la consola, para descubrir que en realidad lo que phpmyadmin hace es convertir lo que le pedimos en sentencias SQL o queries que envía a MySQL

Consola

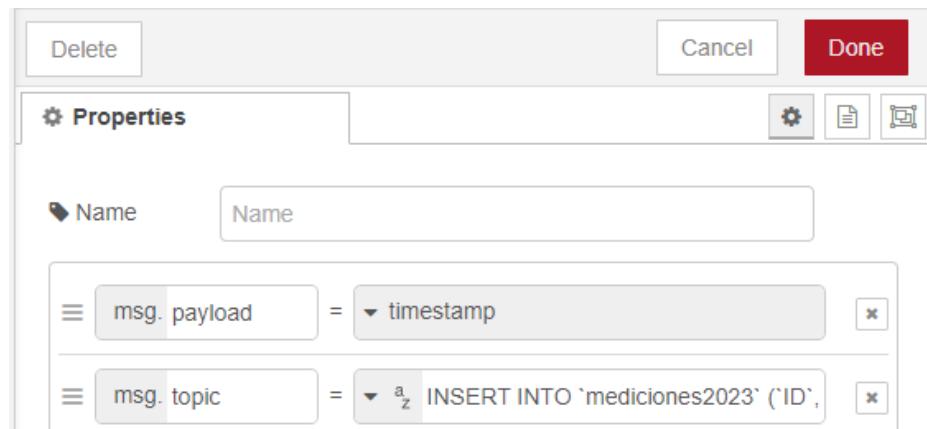
```
>CREATE TABLE `DB_CURSO_IOT`.`mediciones2023` ( `ID` INT NOT NULL AUTO_INCREMENT , `Fecha` DATETIME  
>SELECT * FROM `mediciones2023`  
>INSERT INTO `mediciones2023` (`ID`, `Fecha`, `Dispositivos`, `Temperatura`, `Humedad`) VALUES (  
> |
```

- En conclusión, la sentencia query para configurar el bloque Inject es:

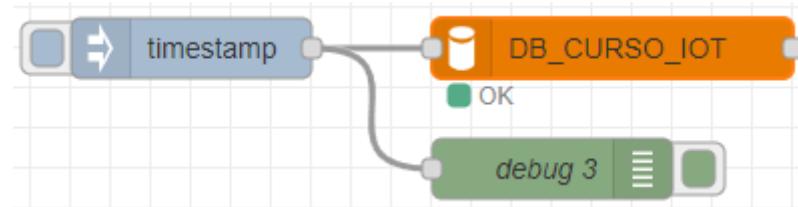
```
INSERT INTO `mediciones2023` (`ID`, `Fecha`, `Dispositivos`,  
`Temperatura`, `Humedad`) VALUES (NULL, CURRENT_TIMESTAMP,  
'ESP32', '20.5', '89.3');
```

- Configuramos el bloque Inject con la sentencia query

Vamos a Node-RED > abrimos el bloque Inject (marcado como Timestamp) > en msg.topic pegamos la anterior sentencia SQL y oprimimos “Done” > Deploy



Como resultado tenemos que la base de datos está conectada porque hay un “connected” debajo del bloque DB_Grupo_B_IOT



- Realización de las pruebas: Para cada prueba hacemos esto:
 - Oprimimos el botón izquierdo del bloque Inject (el marcado como timestamp)
 - Vamos a debug messages con  y observamos los datos generados por el bloque Inject


```
payload: object
topic: "INSERT INTO
`mediciones2023` (`ID`, `Fecha`,
`Dispositivos`, `Temperatura`,
`Humedad`) VALUES (NULL,
CURRENT_TIMESTAMP, 'ESP32', '20.5',
'89.3');"
```
 - Vamos a phpmyadmin para ver qué se ha registrado en la base de datos. Lo hacemos así: refrescamos la página de phpmyadmin > seleccionas tu base de datos > seleccionas tu tabla > Examinar. Debemos ver diferentes registros, uno por cada vez que fue oprimido el botón izquierdo del bloque Inject

Examinar Estructura SQL Buscar Insertar Exportar Imp

Mostrando filas 0 - 4 (total de 5, La consulta tardó 0.0004 segundos.)

```
SELECT * FROM `mediciones2023`
```

Perfilando [Editar en línea] [Editar] [Explicar SQL] [Crear código PHP] [Actualizar]

Mostrar todo Número de filas: 25 Filtrar filas: Buscar en esta tabla Sort by

+ Opciones

	ID	Fecha	Dispositivos	Temperatura	Humedad
<input type="checkbox"/>	1	2023-03-05 12:57:02	ESP32	20.5	89.3
<input type="checkbox"/>	2	2023-03-05 14:14:13	ESP32	20.5	89.3
<input type="checkbox"/>	3	2023-03-05 14:14:32	ESP32	20.5	89.3
<input type="checkbox"/>	4	2023-03-05 14:18:19	ESP32	20.5	89.3
<input type="checkbox"/>	5	2023-03-05 14:20:56	ESP32	20.5	89.3

- Podemos borrar los registros hechos en MySQL con las pruebas anteriores, si no queremos volver a verlos

Inicialmente se insertan datos en la tabla creada **mediciones2024** de forma manual con el fin de conocer la forma (formato) en la que se reciben los datos:

No seguro | 172.17.1.224.96/phpmyadmin/index.php?route=/table/change&db=DB_Grupo_B_IOT&table=mediciones2024 "dagd_v2"

Examinar Estructura SQL Buscar Insertar Exportar Importar Privilegios Operaciones Seguimiento Disparadores

✓ 1 fila insertada.
La Id de la fila insertada es: 1

```
INSERT INTO `mediciones2024` (`ID`, `Fecha`, `Dispositivos`, `Temperatura`, `Humedad`, `Distancia`) VALUES (NULL, '2024-11-15 06:53:52.000000', 'ESP32', '20.5', '40', '24');
```

[Editar en línea] [Editar] [Crear código PHP]

Ejecutar la(s) consulta(s) SQL en la tabla DB_Grupo_B_IOT.mediciones2024: ↴

```
1 INSERT INTO `mediciones2024` (`ID`, `Fecha`, `Dispositivos`, `Temperatura`, `Humedad`, `Distancia`) VALUES (NULL, '2024-11-15 06:53:52.000000', 'ESP32', '20.5', '40', '24');
```

ID	Fecha	Dispositivos	Temperatura	Humedad	Distancia
----	-------	--------------	-------------	---------	-----------

Acá se observa la sentencia query para poder cargar los datos desde Node-RED en la Base de Datos:

Consola Presione Ctrl+Enter para ejecutar la consulta

```
> INSERT INTO `mediciones2024` (`ID`, `Fecha`, `Dispositivos`, `Temperatura`, `Humedad`, `Distancia`) VALUES (NULL, CURRENT_TIMESTAMP, 'ESP32', '18.8', '35', '20');
> INSERT INTO `mediciones2024` (`ID`, `Fecha`, `Dispositivos`, `Temperatura`, `Humedad`, `Distancia`) VALUES (NULL, CURRENT_TIMESTAMP, 'ESP32', '18.8', '35', '20');
```

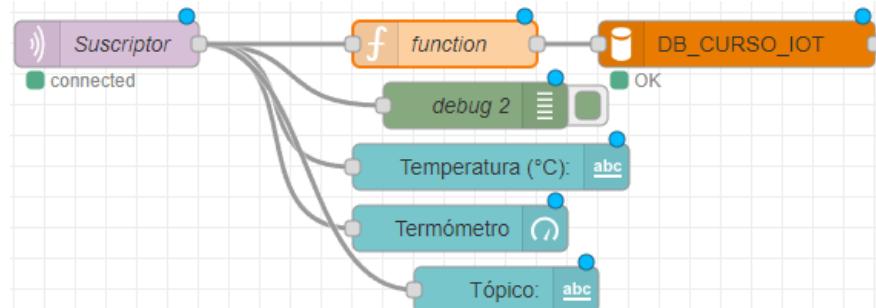
Luego, se observan los datos cargados exitosamente en la Base de Datos:

The screenshot shows two windows side-by-side. On the left is the Node-RED interface with a flow consisting of a timestamp node, a DB node (connected to 'DB_CURSO_IOT'), and a debug 2 node. A 'debug' tab is open, displaying the raw JSON payload sent to the database. On the right is a MySQL database browser window showing the 'mediciones2024' table with four rows of data. The table has columns: ID, Fecha, Dispositivos, Temperatura, Humedad, and Distancia.

ID	Fecha	Dispositivos	Temperatura	Humedad	Distancia
1	2024-11-15 07:53:54	ESP32	18.8	35	20
2	2024-11-15 08:03:34	ESP32	18.8	35	20
3	2024-11-15 08:01:35	ESP32	28.8	45.2	21
4	2024-11-15 08:03:26	ESP32	28.8	45.2	20

Tarea 7. Sprint 1. Uso de la ESP32 pero sin agregar el nuevo sensor

El sub problema a resolver. En la práctica anterior ya se implementó una solución de publicador basado en la ESP32. Volviendo al modelo de la Fig. 2 y basado en los últimos resultados, vemos la necesidad de implementar un bloque a la entrada del DB_Grupo_B_IOT para que prepare la query que requiere ese bloque, entonces a nivel del servidor Node-RED la solución es la siguiente



Donde:

- A diferencia de la Figura 2, no se incluyen los bloques remotos. El Mosquito tampoco porque está implícito en el suscriptor. Entonces tenemos solo los bloques del receptor, con la única diferencia que aparece el bloque “function”
- El bloque “function” tiene la capacidad de embeber código. En otras palabras, puede hacer todo aquello que le programemos internamente
- Lo que necesitamos programar el bloque “function” es que tome los datos que le entrega el suscriptor, que están en JSON y los convierta a la query que requiere DB_Grupo_B_IOT. veamos qué tenemos

Ejemplo del JSON que llega de la ESP32

```
Object {
topic: "canal"
payload: 25.89
```

```

qos: 0
retain: false
_topic: "canal"
msgid: "0891380059916081"
}

```

Ejemplo del query que requiere el bloque DB_Grupo_B_IOT

```

INSERT INTO `mediciones2023` (`ID`, `Fecha`, `Dispositivos`, `Temperatura`,
`Humedad`) VALUES (NULL, CURRENT_TIMESTAMP, 'ESP32', '20.5', '89.3');

```

Programación para el bloque “function”. Nota: como no tenemos aún sensor de humedad, dejamos para ese campo lo mismo que traímos de la tarea anterior.

```

var query = "INSERT INTO `mediciones2023` ('ID', 'Fecha', 'Dispositivos', 'Temperatura', 'Humedad')  

VALUES (NULL, CURRENT_TIMESTAMP, 'ESP32', "",  

query = query + msg.payload + "", '89.3');"  

msg.topic=query;  

return msg;

```

Análisis: el código no hace otra cosa que reemplazar el valor '20.5' por el valor que suministra el ESP32 y que viene en el parámetro msg.payload

Paso 1. Implementamos la solución para el sprint 1

- Reprogramamos la ESP con el software de la práctica anterior, solo lo hacemos con el fin actualizar la IP de la VM
- En Node-RED creamos el esquema anterior, programamos el bloque function > Done > Deploy
- Vamos a phpmyadmin para observar si los datos que entrega el ESP32 están siendo registrados en la base de datos. Apreciamos algo así en la tabla:

SELECT * FROM `mediciones2023`								
<input type="checkbox"/> Perfilando [Editar en línea] [Editar] [Explicar SQL] [Crear código PHP] [Actualizar]								
	1	>	>>	<input type="checkbox"/> Mostrar todo	Número de filas: 25	Filtrar filas: <input type="text"/> Buscar en es		
+ Opciones								
<input type="checkbox"/>	<input type="button" value="←"/>	<input type="button" value="→"/>	ID	Fecha	Dispositivos	Temperatura	Humedad	
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copiar"/>	<input type="checkbox"/>	Borrar 1	2023-03-05 12:57:02	ESP32	20.5	89.3
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copiar"/>	<input type="checkbox"/>	Borrar 2	2023-03-05 14:14:13	ESP32	20.5	89.3
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copiar"/>	<input type="checkbox"/>	Borrar 3	2023-03-05 14:14:32	ESP32	20.5	89.3
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copiar"/>	<input type="checkbox"/>	Borrar 4	2023-03-05 14:18:19	ESP32	20.5	89.3
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copiar"/>	<input type="checkbox"/>	Borrar 5	2023-03-05 14:20:56	ESP32	20.5	89.3
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copiar"/>	<input type="checkbox"/>	Borrar 6	2023-03-05 19:28:43	ESP32	25.08	89.3
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copiar"/>	<input type="checkbox"/>	Borrar 7	2023-03-05 19:28:46	ESP32	25.64	89.3
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copiar"/>	<input type="checkbox"/>	Borrar 8	2023-03-05 19:28:49	ESP32	25.48	89.3

Primero se configura el fluograma (bloque function) para que funcione correctamente el bloque de MySQL:

```

var query = "INSERT INTO `mediciones2024` (`ID`, `Fecha`, `Dispositivos`, `Temperatura`, `Humedad`, `Distancia`) VALUES (NULL,
CURRENT_TIMESTAMP, 'ESP32', '', '';

```

```

query = query + msg.payload.Temperatura + ",'" + "";
query = query + msg.payload.Humedad + ",'" + "";
query = query + msg.payload.Distancia + "')");";
msg.topic = query;
return msg;

```

Esa es la sentencia query para pasar los datos recibidos a formato Json y así poder ser cargados correctamente a la base de datos:

The screenshot displays two windows. The top window is Node-RED, showing a flow where data from various sensors (DHT11, HC-SR04) and a DB node is processed through functions and debug nodes to format JSON messages. The bottom window is a MySQL database interface showing the 'mediciones2024' table with sensor data. To the right, a dashboard titled 'Practica 5: Grupo B' displays real-time sensor values for DHT11 (Temperature 28°C, Humidity 71%) and HC-SR04 (Distance 43 cm).

Node-RED Flow Details:

- Generar Datos** node feeds into **Suscriptor** and **function 1**.
- Suscriptor** feeds into **debug Temp** and **DB_Grupo_B_IOT**.
- function 1** feeds into **debug /**.
- DB_Grupo_B_IOT** feeds into **DB**.
- DB** has an **OK** status.
- debug Temp** and **debug /** both feed into **Temperatura (°C)**, **Humedad (%)**, **Distancia (cm)**, **Tópico abc**, **ID abc**, and **Dispositivo abc** nodes.
- debug /** also feeds into **Object** nodes which log the JSON message to the terminal.

MySQL Database Results:

ID	Fecha	Dispositivos	Temperatura	Humedad	Distancia
1	2024-11-15 07:53:04	ESP32	18.8	35	20
2	2024-11-15 08:00:34	ESP32	18.8	35	20
3	2024-11-15 08:01:35	ESP32	28.8	45.2	21
4	2024-11-15 08:03:26	ESP32	28.8	45.2	20
5	2024-11-15 08:21:17	ESP32	28	72	43
6	2024-11-15 08:21:20	ESP32	28	72	43
7	2024-11-15 08:21:23	ESP32	28	72	34
8	2024-11-15 08:21:26	ESP32	28	72	43
9	2024-11-15 08:21:29	ESP32	28	72	43
10	2024-11-15 08:21:32	ESP32	28	72	43
11	2024-11-15 08:21:35	ESP32	28	72	43
12	2024-11-15 08:21:38	ESP32	28	72	43
13	2024-11-15 08:21:41	ESP32	28	72	34
14	2024-11-15 08:21:44	ESP32	28	72	43
15	2024-11-15 08:21:47	ESP32	28	72	43
16	2024-11-15 08:21:50	ESP32	28	72	43
17	2024-11-15 08:21:53	ESP32	28	72	43
18	2024-11-15 08:21:56	ESP32	28	72	43
19	2024-11-15 08:21:59	ESP32	28	72	43
20	2024-11-15 08:22:02	ESP32	28	72	43
21	2024-11-15 08:22:05	ESP32	28	72	43
22	2024-11-15 08:22:08	ESP32	28	72	43
23	2024-11-15 08:22:11	ESP32	28	72	43
24	2024-11-15 08:22:14	ESP32	28	72	43
25	2024-11-15 08:22:17	ESP32	28	72	43

Dashboard Results:

- Sensor DHT11**: Temperatura (°C): 28, Humedad (%): 71
- Sensor HC-SR04**: Distancia (cm): 43

Tarea 8. Sprint 2. Uso de la ESP32 pero imitando el segundo sensor en la ESP32

El problema a resolver:

Si la ESP32 debe enviar más de un dato en el parámetro payload, vamos a tener que representar payload como en un formato JSON por ejemplo:

JSON que está generando la ESP32 con un sensor

```
Object {  
  topic: "canal"  
  payload  
  qos: 0  
  retain: false  
  _topic: "canal"  
  _msgid: "0891380059916081"  
}
```

JSON que debería general la ESP32 con dos sensores y un parámetro que identifica al dispositivo

```
Object {  
  topic: "canal"  
  payload {  
    Dispositivos: "ESP32"  
    Temperatura: 20.5  
    Humedad: 89.3  
  }  
  qos: 0  
  retain: false  
  _topic: "canal"  
  _msgid: "0891380059916081"  
}
```

Análisis: En el código de la ESP32 es necesario obtener payload en formato JSON. También será necesario reconfigurar el bloque “function” en Node-RED debido al cambio de formato de los datos entrantes.

Paso 1. Identificar el código de serialización (osea, el de pasar datos a JSON)

- Hacer serialización es lo mismo que convertir una información a un objeto JSON. Deserialización es lo contrario
- Con ayuda de <https://arduinojson.org/> descubrimos cuales comandos agregar así:
Assistant >



- Continuamos con “Next JSON” y ajustamos a nuestras necesidades el JSON sugerido

Step 2: JSON

Examples: [OpenWeatherMap](#), [Reddit](#)

Output

Enter here the JSON document you want your program to generate.

```
{  
    "Dispositivos": "ESP32",  
    "Temperatura": 25,  
    "Humedad": 70  
}
```

- Continuamos con “Next JSON” > “Next JSON”. Como resultado nos entrega el código que hay que adicionar en el Arduino IDE

En realidad, para poder usar ese código es necesario incluir librerías.

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

El siguiente es el código sugerido por <https://arduinojson.org/>. Debe ser ubicado como parte de la función void loop()

```
StaticJsonDocument<64> doc;  
  
doc["Dispositivos"] = "ESP32";  
doc["Temperatura"] = 25;  
doc["Humedad"] = 70;  
  
serializeJson(doc, output);
```

Paso 2. Complementación del código de la ESP32 con el de serialización

```
// Librerias a usar  
#include "WiFi.h"  
#include "PubSubClient.h"  
#include <ArduinoJson.h>  
  
// Resolucion deseada en las mediciones  
#define ADC_VREF_mV 3300.0  
#define ADC_RESOLUTION 4096.0  
  
// Datos de la WiFi  
WiFiClient esp32Client;  
const char* ssid = "CHAF";  
const char* password = "chaf2002";  
  
// Datos Broker y clientes  
PubSubClient mqttClient(esp32Client);  
char* server = "35.153.57.109"; // La IP pública de la VM en AWS  
int port = 1883; // el puerto que tiene Mosquitto asignado, por defecto 1883  
char datos[40]; // variable usada para los valores de las mediciones  
  
// Datos de la ESP32  
int temp = 33; // el pin al cual se conecta el sensor  
int led = 2; // el led 2 (azul) será usado para mostrar que la WiFi está conectada  
int boton = 0; // el botón que usaremos para resetear la tarjeta  
  
//-----  
void wifi() {
```

```

Serial.print("Conectandose a WiFi: ");
Serial.println(ssid); // el nombre del usuario de WiFi

WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) { //mostrar peticos mientras no logra conexion
    Serial.print(".");
    delay(500);
}
digitalWrite(led, HIGH);

//Serial.println("");
Serial.println("\nConectado a WIFI");
Serial.print("Dirección IP ");
Serial.println(WiFi.localIP());
}

//-----
// si se pierde la conexion MQTT esta función será usada para reconectarse
void reconnect() {
    while (!mqttClient.connected()) {
        Serial.print("Conexion MQTT perdida, intentando conectarse ...");
        if (mqttClient.connect("esp32")) {
            Serial.println("Conectado");
        } else {
            Serial.print("Falló, rc=");
            Serial.print(mqttClient.state());
            Serial.println(" intentar de nuevo en 5 segundos");
            delay(5000); // espera de 5 segundos
        }
    }
}

void setup() {
    // Inicialización de los pines
    pinMode(led, OUTPUT); // el pin del LED se configura como una salida
    pinMode(boton, INPUT); // el pin botón como una entrada
    Serial.begin(115200); // abrir el monitor serial del computador a esa rata
    delay(10);
    wifi(); // ordena la conexión a wifi
    mqttClient.setServer(server, port); // ordena la conexión al broker
}

//-----
void loop() {
    // Si se pierde la conexión WiFi, recuperarla
    if (WiFi.status() != WL_CONNECTED) {
        Serial.println("Conexion WiFi perdida. Intentando reconectar...");
        digitalWrite(led, LOW);
        wifi(); // intentar restablecer la conexión
    }

    // Si se pierde la conexión con el servidor, recuperarla
    if (!mqttClient.connected()) {
        reconnect();
    }
    // Captar y publicar los datos del sensor de temperatura
    float temperatura = (analogRead(temp) * (ADC_VREF_mV / ADC_RESOLUTION)) / 10.0 + 8;
    StaticJsonDocument<64> doc;
    doc["Dispositivos"] = "ESP32";
    doc["Temperatura"] = temperatura;
    doc["Humedad"] = random(60, 90);
    String datos;
    serializeJson(doc, datos);
}

```

```

Serial.print("Datos enviados: ");
Serial.println(datos);

mqttClient.publish("canal", datos.c_str());
delay(3000);
}

```

Análisis

- En el código sugerido hemos cambiado “output” por “datos” que es nuestra variable
- tuvimos que declarar “datos” como de tipo String
- usamos datos.c_str() porque en general lo que hay que enviar ahora es un string.

Paso 3. Pruebas a nivel de Node-RED de la operación de serialización

- Implementamos el flujograma más sencillo porque, de otra manera, será necesario reconfigurar todos los bloques. Además porque solo nos interesa comprobar qué llega



- Bajamos a la tarjeta ESP32 el software actualizado y revisamos que debug nos entrega datos como estos, que corresponden a lo esperado:

```

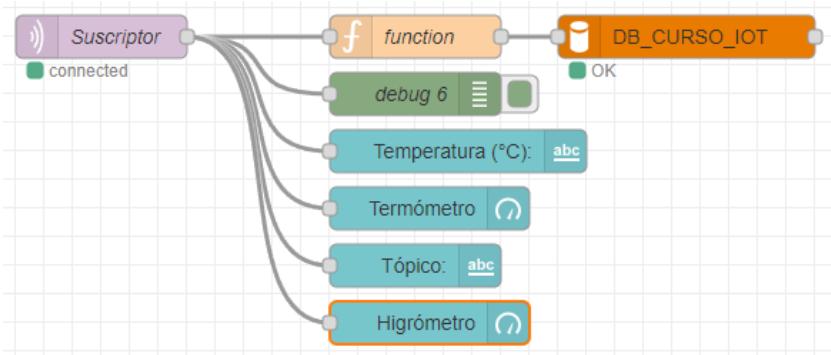
3/5/2023, 5:17:52 PM node: debug 5
canal : msg : Object
  ▼ object
    topic: "canal"
    ▼ payload: object
      Dispositivos: "ESP32"
      Temperatura: 26.93310547
      Humedad: 65
      qos: 0
      retain: false
      _topic: "canal"
      _msgid: "8aec843c5ca276e8"

```

Paso 4. Montaje final para el sprint 2

Considerando:

- De la tarea anterior, ya traímos un flujograma que respondía bien a los intereses de este desarrollo. Pero no nos satisfacía la manera en que llegaban los datos
- Como ahora, la carga útil está llegando en un formato JSON debemos reconfigurar los bloques
- Agregamos un bloque adicional para graficar la humedad, el Higrómetro



Bloque Function

Consideramos los parámetros que llegan de la ESP32 dentro de payload

```

var query = "INSERT INTO `mediciones2023` (`ID`, `Fecha`, `Dispositivos`, `Temperatura`, `Humedad`) VALUES (NULL, CURRENT_TIMESTAMP, ";
query = query + msg.payload.Dispositivos + ", ";
query = query + msg.payload.Temperatura + ", ";
query = query + msg.payload.Humedad + ");";
msg.topic=query;
return msg;

```

Bloque Temperatura y Termómetro

{{msg.payload.Temperatura}}

Bloque Hidrómetro

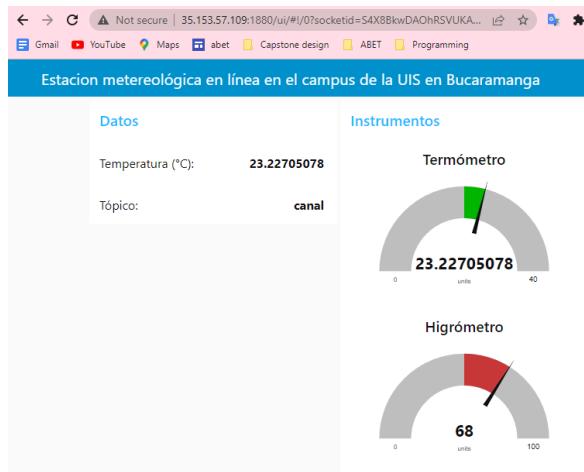
Properties	
Group	[Estacion metereologica en linea en el]
Size	auto
Type	Gauge
Label	Hidrómetro
Value format	{{msg.payload.Humedad}}
Units	units
Range	min 0 max 100

Realizamos observaciones finales:

- A la base de datos para comprobar que si se registran los datos

<input type="checkbox"/>		Editar		Copiar		Borrar	287	2023-03-05 23:03:11	ESP32	25.2412	67
<input type="checkbox"/>		Editar		Copiar		Borrar	288	2023-03-05 23:03:14	ESP32	24.7578	64
<input type="checkbox"/>		Editar		Copiar		Borrar	289	2023-03-05 23:03:17	ESP32	24.6772	88
<input type="checkbox"/>		Editar		Copiar		Borrar	290	2023-03-05 23:03:20	ESP32	24.7578	72
<input type="checkbox"/>		Editar		Copiar		Borrar	291	2023-03-05 23:03:23	ESP32	24.5161	73
<input type="checkbox"/>		Editar		Copiar		Borrar	292	2023-03-05 23:03:26	ESP32	26.0469	70

- A los instrumentos



En la **Tarea 7** ya se realizo eso implícitamente, ya que como se observa en un pantallazo de las evidencias ya se cargan los datos que publica la ESP32 en la Base de Datos, acá se mostrará como se realizó la parte de la modificación del código de la ESP32 en el que se agrego la parte de JSON junto con la librería <ArduinoJson.h>, esto para hacer serialización que es lo mismo que convertir una información a un objeto JSON:

The screenshot shows the ArduinoJson.org v7 assistant interface. At the top, there's a purple header with a back arrow, forward arrow, refresh icon, and a URL field containing "https://arduinojson.org/v7/assistant/#/step2". To the right of the URL are icons for font size, font style, and a star. Below the header, there are three numbered steps: 1 Configuration, 2 JSON (which is highlighted in green), and 3 Program. A teal banner at the top says "Step 2: JSON". Below the banner, it says "Examples: [OpenWeatherMap](#), [Reddit](#)". Under the heading "Output", there is a code editor containing the following JSON object:

```
{
  "Dispositivos": "ESP32",
  "Temperatura": 20.5,
  "Humedad": 33.5,
  "Distancia": 20.8
}
```

```

Step 3: Program

#include <ArduinoJson.h>

JsonDocument doc;

doc["Dispositivos"] = "ESP32";
doc["Temperatura"] = 28.5;
doc["Humedad"] = 33.5;
doc["Distancia"] = 20.8;

String output;

doc.shrinkToFit(); // optional

serializeJson(doc, output);

practica_5.ino
119   digitalWrite(pinLedDistancia, LOW);
120 }
121
122 // Leer la humedad y temperatura del sensor DHT11
123 float h = dht.readHumidity();
124 float t = dht.readTemperature();
125
126 //StaticJsonDocument<64> doc; // Usa DynamicJsonDocument en lugar de StaticJsonDocument<64>
127 //DynamicJsonDocument doc(64); // Usa DynamicJsonDocument en lugar de StaticJsonDocument<64>
128 -> JsonDocument doc;
129 ..doc["Dispositivos"]-=>"ESP32";
130 ..doc["Temperatura"]-=>t;
131 ..doc["Humedad"]-=>h;
132 ..doc["Distancia"]-=>distancia;
133 ..String-datos;
134 ..doc.shrinkToFit();
135 ..serializeJson(doc, datos);
136
137 // Verificar si las lecturas del sensor DHT11 son válidas

Salida Monitor Serie x
Mensaje (Intro para mandar el mensaje al ESP32 Dev Mod. Nueva linea 115200 baud
03:36:50.128 -> {"Dispositivos": "ESP32", "Temperatura": 28, "Humedad": 71, "Distancia": 20cm}
03:36:53.191 -> Humedad: 71.00% Temperatura: 28.00°C Distancia:20cm
03:36:53.191 -> {"Dispositivos": "ESP32", "Temperatura": 28, "Humedad": 71, "Distancia": 20cm}
03:36:56.176 -> Humedad: 71.00% Temperatura: 28.00°C Distancia:20cm
03:36:56.250 -> {"Dispositivos": "ESP32", "Temperatura": 28, "Humedad": 71, "Distancia": 20cm}
03:36:59.231 -> Humedad: 71.00% Temperatura: 28.00°C Distancia:20cm
03:36:59.231 -> {"Dispositivos": "ESP32", "Temperatura": 28, "Humedad": 71, "Distancia": 20cm}

Lin. 128, col. 1 ESP32 Dev Module en COM5 0 2

```

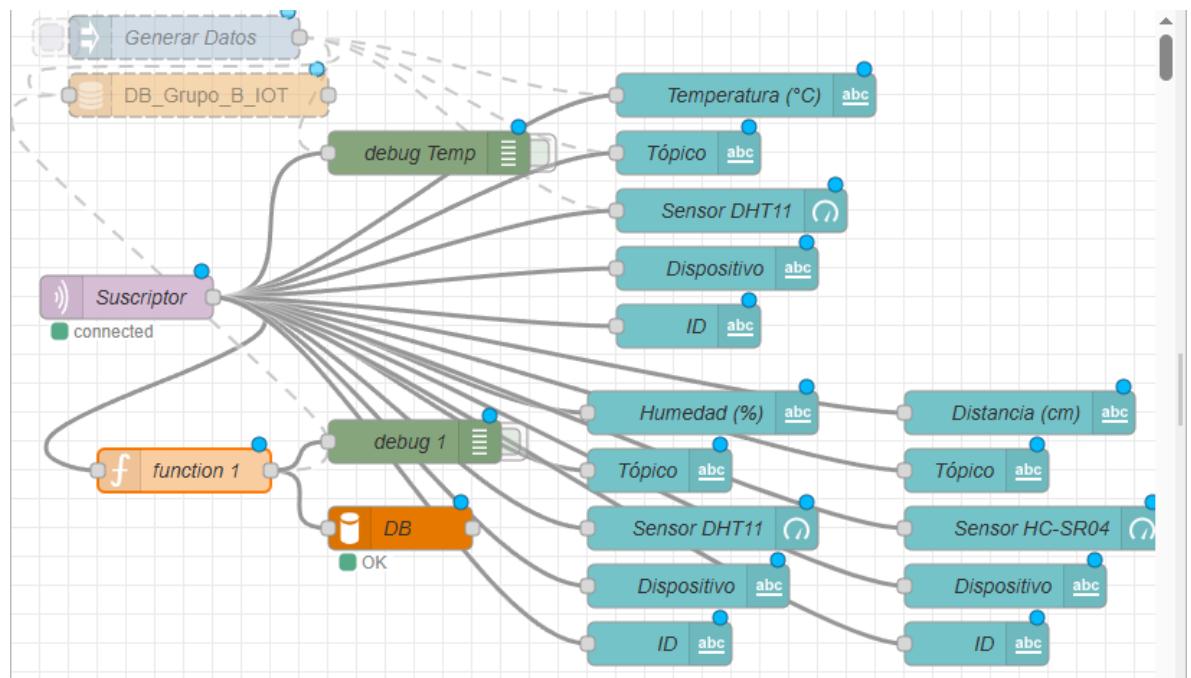
En el bloque **function** se usó la siguiente query:

```

var query = "INSERT INTO `mediciones2024` (`ID`, `Fecha`,
`Dispositivos`, `Temperatura`, `Humedad`, `Distancia`) VALUES (NULL,
CURRENT_TIMESTAMP, '";
query = query + msg.payload.Dispositivos + "', '')";
query = query + msg.payload.Temperatura + "', '')";
query = query + msg.payload.Humedad + "', '')";
query = query + msg.payload.Distancia + "')";

msg.topic = query;
return msg;

```



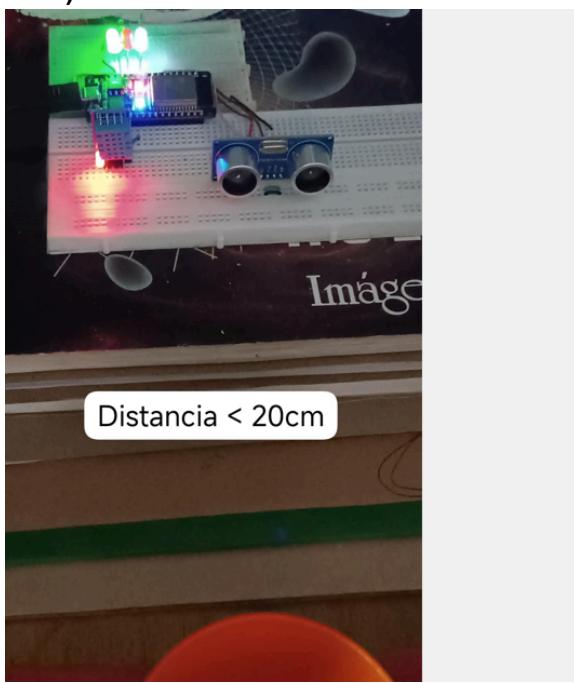
De no usarse ese bloque **function** y no usar la **query** los datos no se cargarían en la Base de Datos, finalmente se muestra que ya hay más de 399 datos cargados en la Base

de Datos, se aclara que están usando dos sensores, uno para temperatura y humedad y, el otro para distancia.

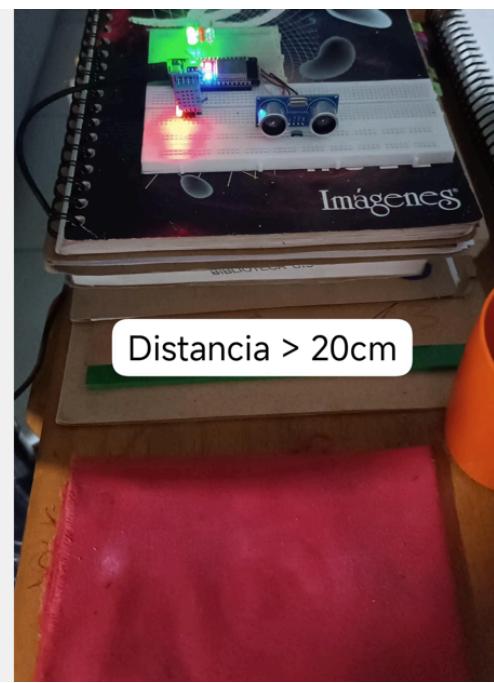
Tarea 9. Sprint 3. Uso de la ESP32 incluyendo el segundo sensor real en la ESP32

Para esta tarea, simplemente se mostrarán pantallazos de todo lo realizado ya que desde el inicio mencioné que iba a usar sensores, es decir desde el inicio realice el desarrollo de esta práctica usando los sensores **DHT11** y **HC-SR04**.

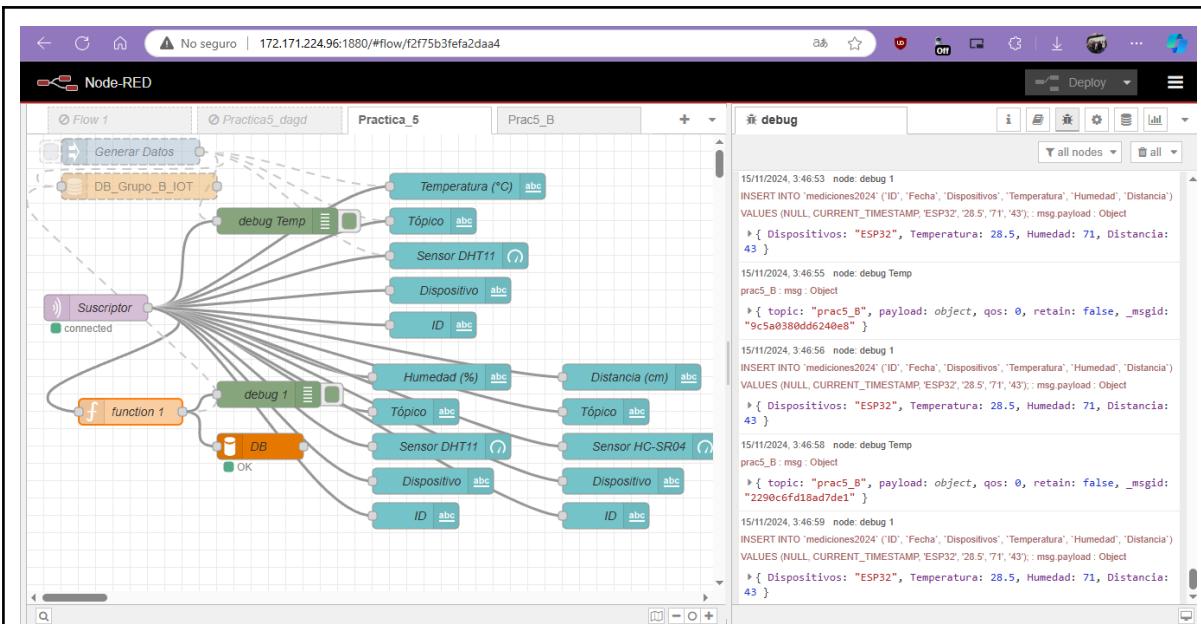
1) Uso de la ESP32 con los sensores y varios leds indicadores:



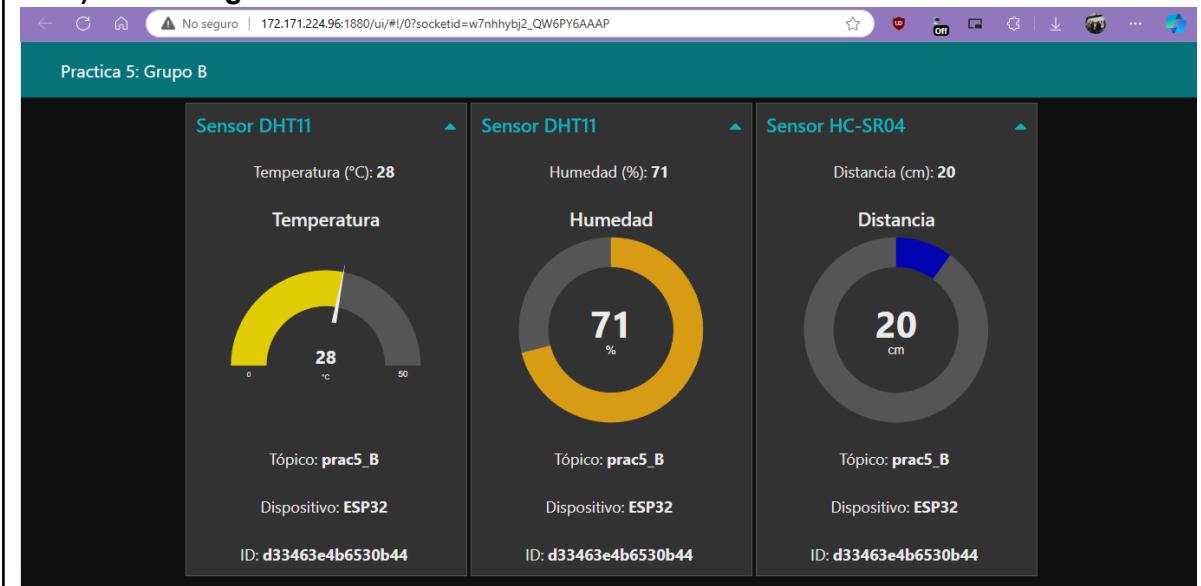
Distancia < 20cm



2) Flujograma realizado en Node-RED:



3) Interfaz gráfica:



4) Datos cargados en la Base de Datos (phpmyadmin):

No seguro | 172.171.224.96/phpmyadmin/index.php?route=/sql&db=DB_Grupo_B_IOT&table=mediciones2024 "dagd_v2"

Servidor: localhost:3306 » Base de datos: DB_Grupo_B_IOT » Tabla: mediciones2024 "dagd_v2"

Examinar Estructura SQL Buscar Insertar Exportar Importar Privilegios O

Mostrando filas 0 - 249 (total de 441, La consulta tardó 0.0004 segundos.) [ID: 441... - 192...]

SELECT * FROM `mediciones2024` ORDER BY `ID` DESC

Perfilando [Editar en línea] [Editar] [Explicar SQL] [Crear código PHP] [Actualizar]

1 > >> Mostrar todo Número de filas: 250 Filtrar filas: Buscar en esta tabla Ordenar según la c

Opciones extra

ID	Fecha	Dispositivos	Temperatura	Humedad	Distancia
441	2024-11-15 08:43:44	ESP32	28	71	20
440	2024-11-15 08:43:41	ESP32	28	71	20
439	2024-11-15 08:43:38	ESP32	28	71	20
438	2024-11-15 08:43:35	ESP32	28	71	20
437	2024-11-15 08:43:32	ESP32	28	71	20
436	2024-11-15 08:43:29	ESP32	28	71	20
435	2024-11-15 08:43:26	ESP32	28	71	20
434	2024-11-15 08:43:23	ESP32	28	71	20
433	2024-11-15 08:43:20	ESP32	28	71	20
432	2024-11-15 08:43:17	ESP32	28	71	20
431	2024-11-15 08:43:14	ESP32	28	71	20
Consola					

5) Código usado en la ESP32:

```
// By dagdmfc

// Librerías a usar
#include <WiFi.h>
#include <PubSubClient.h>
#include <DHT.h>
#include <ArduinoJson.h>

// Definición pines de los sensores (DHT11, HC-SR04 y leds) en la
//ESP32
#define DHTPIN 4           // Pin para el sensor DHT11
#define DHTTYPE DHT11      // Tipo de sensor DHT
DHT dht(DHTPIN, DHTTYPE);
int pinLedDistancia = 27; // LED que enciende cuando la distancia es
menor a 20 cm
int echoPin = 32;         // Pin de Echo del sensor HC-SR04
int triggerPin = 33;      // Pin de Trigger del sensor HC-SR04
int ledWiFi = 2;          // LED indicador de conexión WiFi
int ledMQTTConectado = 13; // LED para indicar conexión MQTT exitosa
int ledMQTTDesconectado = 12; // LED para indicar falla en conexión
MQTT

// WiFi credentials
WiFiClient esp32Client;
//const char* ssid = "DAGDMFC";
//const char* password = "millos2017";
const char* ssid = "PISO_2";
```

```

const char* password = "Planetaland2";

// Datos Broker y clientes
PubSubClient mqttClient(esp32Client);
const char* server = "172.171.224.96"; // La IP pública de la VM en
Azure
//char* server = "broker.emqx.io";
int port = 1883; // El puerto que tiene Mosquitto asignado, por
defecto 1883
char datos[60]; // Variable usada para los valores de las mediciones

// Función para calcular la distancia en cm usando el sensor
ultrásónico
long readUltrasonicDistance(int triggerPin, int echoPin) {
    pinMode(triggerPin, OUTPUT);
    digitalWrite(triggerPin, LOW);
    delayMicroseconds(2);
    digitalWrite(triggerPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(triggerPin, LOW);
    pinMode(echoPin, INPUT);
    return pulseIn(echoPin, HIGH);
}

// Conexión a la WiFi
void wifi() {
    Serial.print("Conectándose a WiFi: ");
    Serial.println(ssid);

    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) { // Mostrar puntitos
mientras no logra conexión
        Serial.print(".");
        delay(500);
    }
    digitalWrite(ledWiFi, HIGH); // Encender el LED al conectar WiFi
    Serial.println("\nConectado a WiFi");
    Serial.print("Dirección IP: ");
    Serial.println(WiFi.localIP());
}

// Reconexión al broker MQTT
void reconnect() {
    while (!mqttClient.connected()) {
        Serial.print("Conexión MQTT perdida, intentando reconectar... ");
        Serial.print(".");
        delay(500);
        if (mqttClient.connect("esp32Client")) {
            Serial.println("Conectado al broker MQTT");
            digitalWrite(ledMQTTConectado, HIGH); // Encender LED MQTT
cuando esté conectado
            digitalWrite(ledMQTTDesconectado, LOW); // Apagar LED de
desconexión
        } else {
            Serial.print("Fallo, rc=");
            Serial.print(mqttClient.state());
            Serial.println(" Intentar de nuevo en 5 segundos");
        }
    }
}

```

```

        Serial.print(".");
        delay(500);
        digitalWrite(ledMQTTConectado, LOW); // Apagar LED MQTT si la
conexión falla
        digitalWrite(ledMQTTDesconectado, HIGH); // Encender LED de
desconexión
        delay(3000); // Espera de 5 segundos
    }
}
}

void setup() {
    // Configuración de pines
    pinMode(ledWiFi, OUTPUT); // LED indicador de WiFi
    pinMode(pinLedDistancia, OUTPUT); // LED indicador de distancia
    pinMode(ledMQTTConectado, OUTPUT); // LED indicador de conexión
MQTT
    pinMode(ledMQTTDesconectado, OUTPUT); // LED indicador de
desconexión MQTT

    // Configuración del monitor serial y del sensor DHT11
    Serial.begin(115200);
    delay(10);
    dht.begin();

    // Conexión a WiFi y configuración del servidor MQTT
    wifi();
    mqttClient.setServer(server, port);
}

void loop() {
    // Si se pierde la conexión WiFi, intentar reconectar
    if (WiFi.status() != WL_CONNECTED) {
        Serial.println("Conexión WiFi perdida. Intentando
reconectar...");
        digitalWrite(ledWiFi, LOW); // Apagar LED WiFi
        wifi(); // Intentar reconectar
    }

    // Si se pierde la conexión con el broker MQTT, intentar reconectar
    if (!mqttClient.connected()) {
        reconnect();
    }

    // Calcular la distancia medida por el sensor ultrasónico
    int distancia = 0.01723 * readUltrasonicDistance(triggerPin,
echoPin);
    // Encender el LED si la distancia es menor a 20 cm
    if (distancia < 20) {
        digitalWrite(pinLedDistancia, HIGH);
    } else {
        digitalWrite(pinLedDistancia, LOW);
    }

    // Leer la humedad y temperatura del sensor DHT11
    float h = dht.readHumidity();
    float t = dht.readTemperature();
}

```

```
//StaticJsonDocument<64> doc;
//DynamicJsonDocument doc(64); // Usa DynamicJsonDocument en lugar
de StaticJsonDocument
JsonDocument doc;
doc["Dispositivos"] = "ESP32";
doc["Temperatura"] = t;
doc["Humedad"] = h;
doc["Distancia"] = distancia;
String datos;
doc.shrinkToFit();
serializeJson(doc, datos);

// Verificar si las lecturas del sensor DHT11 son válidas
if (isnan(h) || isnan(t)) {
    Serial.println(F("Fallo al leer el sensor DHT!"));
} else {
    // Mostrar en el monitor serial
    Serial.print("Humedad: ");
    Serial.print(h);
    Serial.print("% Temperatura: ");
    Serial.print(t);
    Serial.print("°C Distancia:");
    Serial.print(distancia);
    Serial.println("cm");
}
// Publicar la distancia, temperatura y humedad en el broker MQTT
//sprintf(datos, "Distancia: %d, Temp: %.f, Humedad: %.f",
distancia, t, h);
Serial.println(datos);
mqttClient.publish("prac5_B", datos.c_str());
delay(3000); // Espera antes de la siguiente lectura
}
```