PROYECTO: MONITOR DE SENSOR DE PROXIMIDAD

MATERIA: BASES DE DATOS II Y LABORATORIO

ALUMNO: LUIS ANGEL PLASCENCIA SALAZAR

MATRICULA:42202377

FECHA DE ENTREGA :18/02/2025

DOCENTE: ALEJANDRO ULISES LÓPEZ OROZCO

Objetivo.

El objetivo de este proyecto es configurar un ESP32 con detección de proximidad (HC-SR04 o similar) para registrar la distancia de objetos cercanos y enviar los datos a un servidor Node.js que los almacenará en MongoDB Atlas junto con la clave del sensor, la fecha, hora y coordenadas GPS (latitud y longitud).

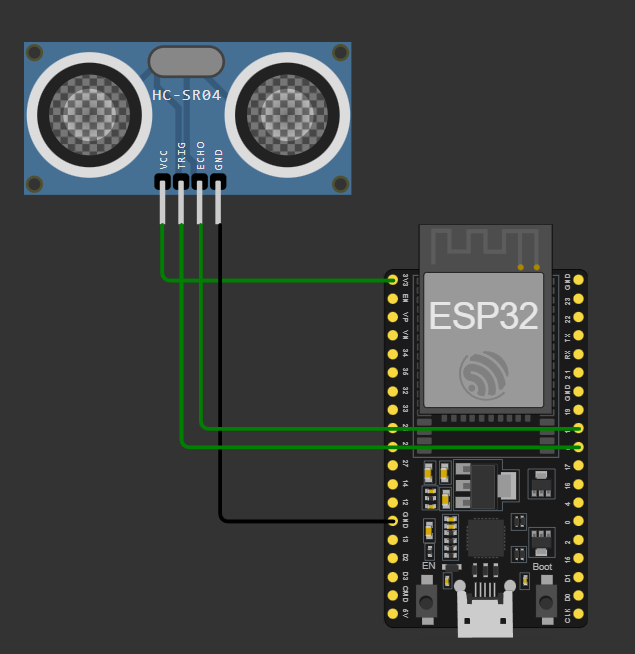
Materiales:

* ESP32 (placa de desarrollo con WiFi y Bluetooth)
* Sensor de Ultrasonido HC-SR04
* Cables Jumper (macho-macho, macho-hembra)
* Protoboard (opcional, facilita las conexiones)
* Fuente de alimentación (puedes alimentar el ESP32 con USB)

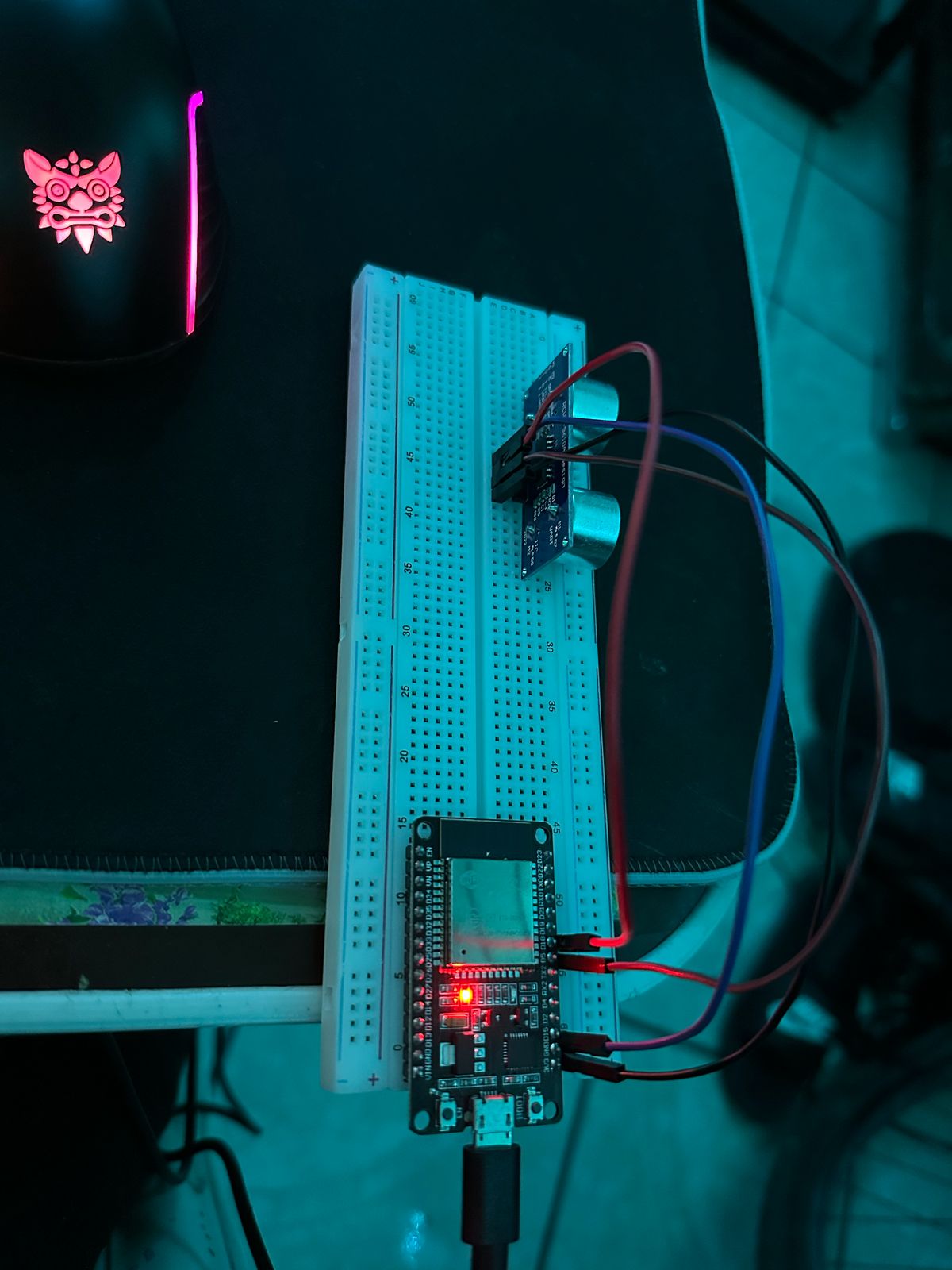
Metodología.

Paso 1.

Lo primero que hice fue en wokwi crear un esquema de conexión de cómo se miraría ya conectado a la protoboard.



En la Protoboard:



Paso 2.

En Arduino ide lo que hice fue crear el código para la configuración de la esp32

Código:

#include <WiFi.h> // Librería para gestionar la conexión WiFi en el ESP32

#include <HTTPClient.h> // Librería para realizar solicitudes HTTP

#include <Ultrasonic.h> // Librería para manejar el sensor ultrasónico HC-SR04

#include "time.h" // Para sincronizar con NTP

// Configuración de la red WiFi a la que se conectará el ESP32

const char\* ssid = "TUUNET-FO-3548"; // Nombre de la red WiFi

const char\* password = "P3043BAR7"; // Contraseña de la red WiFi

// URL del servidor Node.js al que se enviarán los datos

const char\* serverUrl = "<http://192.168.100.8:3000/api/sensor>";

// Definición de los pines utilizados por el sensor ultrasónico HC-SR04

#define TRIG\_PIN 5 // Pin de disparo (Trigger)

#define ECHO\_PIN 18 // Pin de eco (Echo)

// Creación del objeto sensor con los pines configurados

Ultrasonic sensor(TRIG\_PIN, ECHO\_PIN);

// Servidor NTP (puedes usar "pool.ntp.org" u otro)

const char\* ntpServer = "pool.ntp.org";

// Ajuste de zona horaria (ejemplo: México central -6 horas)

const long gmtOffset\_sec = -6 \* 3600; // UTC-6

const int daylightOffset\_sec = 0; // Sin horario de verano

void setup() {

Serial.begin(115200); // Inicia la comunicación serie a 115200 baudios

// Conexión a la red WiFi

WiFi.begin(ssid, password);

Serial.print("Conectando a WiFi...");

// Esperar hasta que el ESP32 se conecte al WiFi

while (WiFi.status() != WL\_CONNECTED) {

delay(1000);

Serial.print(".");

}

Serial.println("\nConectado a WiFi");

// Configurar el NTP

configTime(gmtOffset\_sec, daylightOffset\_sec, ntpServer);

// Esperar un momento para que se sincronicen la fecha y hora

delay(2000);

// Comprobación opcional de la hora actual

struct tm timeinfo;

if (!getLocalTime(&timeinfo)) {

Serial.println("Error al obtener la hora");

} else {

Serial.print("Fecha/Hora actual: ");

Serial.println(&timeinfo, "%Y-%m-%d %H:%M:%S");

}

}

void loop() {

// Medición de la distancia utilizando el sensor ultrasónico

float distancia = sensor.read();

Serial.print("Distancia: ");

Serial.println(distancia); // Imprime la distancia medida en cm

// Solo enviamos los datos si la distancia es mayor a 0 y menor a 50

if (distancia > 0 && distancia <50) {

// Verifica si el ESP32 sigue conectado a la red WiFi

if (WiFi.status() == WL\_CONNECTED) {

HTTPClient http; // Crea un objeto para realizar solicitudes HTTP

http.begin(serverUrl); // Especifica la URL del servidor

http.addHeader("Content-Type", "application/json"); // Especifica el tipo de contenido como JSON

// Construcción del JSON con los datos a enviar

String payload = "{";

payload += "\"sensorKey\": \"ESP32\_001\","; // Identificador del sensor

payload += "\"distance\": " + String(distancia) + ","; // Valor de la distancia medida

payload += "\"timestamp\": \"" + getFormattedTime() + "\","; // Fecha y hora actual

payload += "\"latitude\": 19.4326,"; // Coordenada de latitud (ejemplo)

payload += "\"longitude\": -99.1332"; // Coordenada de longitud (ejemplo)

payload += "}";

// Envío de la solicitud HTTP POST al servidor

int httpResponseCode = http.POST(payload);

Serial.println("Enviando datos...");

Serial.println(payload); // Muestra el JSON que se envía al servidor

// Verifica si el envío fue exitoso

if (httpResponseCode > 0) {

Serial.println("Datos enviados con éxito: " + String(httpResponseCode));

} else {

Serial.println("Error en el envío: " + String(httpResponseCode));

}

http.end(); // Finaliza la conexión HTTP

} else {

Serial.println("Error: No conectado a WiFi"); // Mensaje de error si la conexión falla

}

} else {

Serial.println("Distancia inválida, no se enviarán datos.");

}

delay(5000); // Esperar 5 segundos antes de la siguiente medición

}

// Función para obtener la fecha y hora actual (simulada, sin RTC ni NTP)

String getFormattedTime() {

time\_t now = time(nullptr); // Obtiene el tiempo actual en segundos desde 1970

struct tm\* t = localtime(&now); // Convierte el tiempo a formato de fecha y hora

char buffer[30];

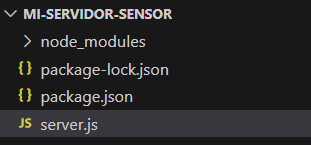
strftime(buffer, sizeof(buffer), "%Y-%m-%d %H:%M:%S", t); // Formatea la fecha y hora

return String(buffer); // Devuelve la fecha y hora como un string

}

Paso 3.

Lo siguiente que hice fue crear nuestro archivo .js que lo llamé server.js, para ello desde nuestros archivos primero crearemos una carpeta en este caso lo llamé mi-server-sensor, esta carpeta la abriremos desde VisualStudio y ahí crearemos nuestro server.js



Ahí introduciremos el código que nos permite recibir los datos del esp32, conectarnos a nuestro Mongodb Altas crear el servidor en el puerto 3000 y definir el esquema y modelo para la conexión.

Código:

// Importamos las librerías necesarias

const express = require("express"); // Framework para manejar peticiones HTTP

const mongoose = require("mongoose"); // Librería para conectarse y manejar MongoDB

const cors = require("cors"); // Middleware para permitir solicitudes desde otros dominios (CORS)

// Conectar a MongoDB Atlas y seleccionar la base de datos ProyectoSensor

mongoose.connect("mongodb+srv://42202377:luan2004plsa@luis.v46eh.mongodb.net/?retryWrites=true&w=majority&appName=Luis", {

useNewUrlParser: true, // Usa el nuevo parser de URL de MongoDB

useUnifiedTopology: true // Usa el nuevo motor de monitoreo de servidores

})

.then(() => console.log("Conectado a MongoDB Atlas")) // Mensaje si la conexión es exitosa

.catch((err) => console.error("Error al conectar a MongoDB:", err)); // Mensaje en caso de error

// Creamos una aplicación Express

const app = express();

// Middleware para permitir solicitudes desde otros dominios (CORS)

app.use(cors());

// Middleware para procesar datos en formato JSON en las peticiones HTTP

app.use(express.json());

// Definir el esquema (estructura) de los documentos en la colección SensorProximidad

const sensorSchema = new mongoose.Schema({

sensorKey: String, // Identificador del sensor

distance: Number, // Distancia medida por el sensor

timestamp: String, // Marca de tiempo del dato recibido

latitude: Number, // Coordenada de latitud

longitude: Number // Coordenada de longitud

});

// Definir el modelo basado en el esquema para interactuar con MongoDB

// El tercer parámetro "SensorProximidad" especifica el nombre exacto de la colección en la base de datos

const SensorProximidad = mongoose.model("SensorProximidad", sensorSchema, "SensorProximidad");

// Ruta para recibir datos del ESP32 mediante una solicitud HTTP POST

app.post('/api/sensor', async (req, res) => {

// Extraer datos del cuerpo de la petición

const { sensorKey, distance, timestamp, latitude, longitude } = req.body;

// Crear una nueva instancia del modelo con los datos recibidos

const sensorData = new SensorProximidad({

sensorKey,

distance,

timestamp,

latitude,

longitude

});

try {

// Guardar los datos en la base de datos

const result = await sensorData.save();

console.log('Documento insertado:', result);

res.status(201).send('Datos recibidos y guardados correctamente'); // Respuesta al cliente

} catch (error) {

console.error('Error al guardar los datos:', error);

res.status(500).send('Error al guardar los datos'); // Respuesta en caso de error

}

});

// Definir el puerto donde escuchará el servidor

const PORT = 3000;

app.listen(PORT, () => {

console.log(`Servidor corriendo en http://localhost:${PORT}`); // Mensaje en consola al iniciar el servidor

});

Paso 4.

Base de datos en Mongodb Atlas

También es importante tener creada nuestra base de datos.

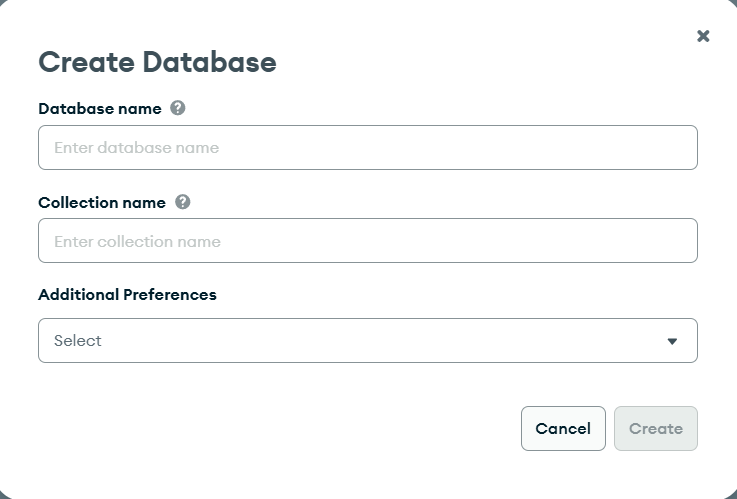
Para ello:

* Ingresa a [MongoDB Atlas Database | Multi-Cloud Database Service | MongoDB](https://www.mongodb.com/products/platform/atlas-database)
* Inicia sesión o crea una
* Da clic en create



* Y sigue los pasos.
* Ahora ya podrás crear una base de datos y una coleccion en



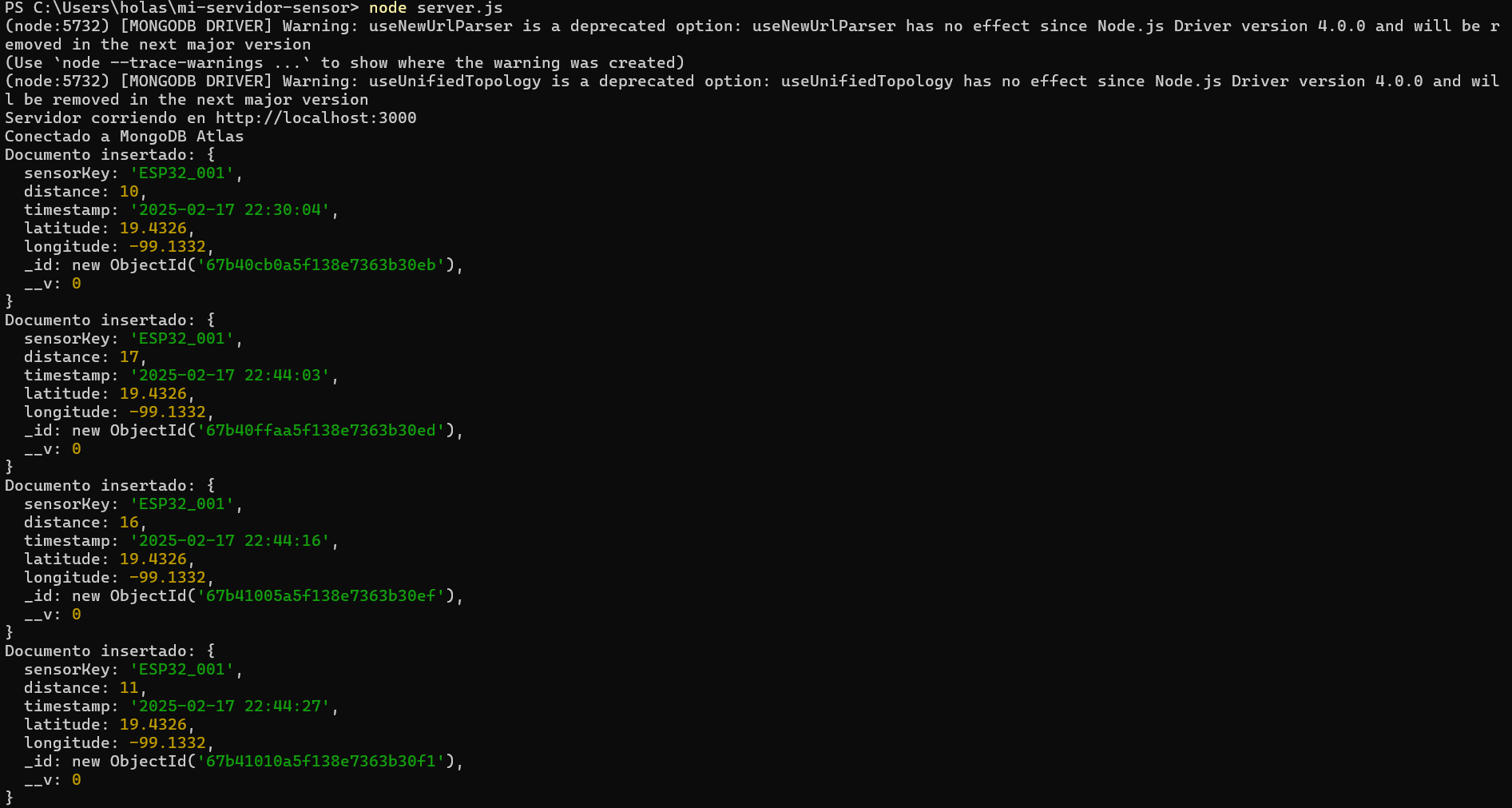


Paso 5.

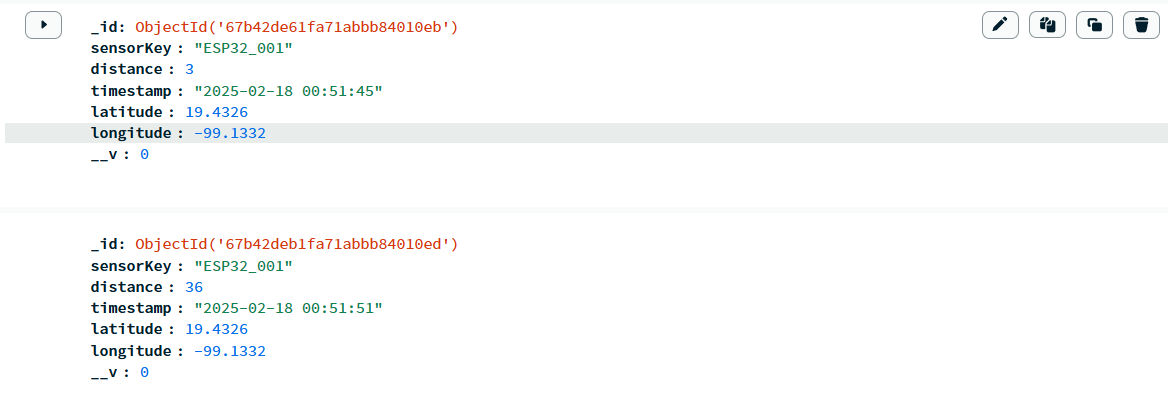
Una vez creada nuestra base de datos lo siguiente será verificar que funcione para ello nos iremos a PowerShell o Cmd y escribiremos lo siguiente.



Si todo va bien nos saldría nuestros documentos agregándose (Nota: si hace falta descargar unas dependencias instálalas).



También verificamos que se hayan agregado a nuestra base de datos



Conclusión

En este proyecto, aprendimos a enviar datos desde un ESP32 a un servidor Node.js y almacenarlos en una base de datos, en este caso, MongoDB Atlas. En general aprendimos sobre la comunicación entre dispositivos y servidores, así como la integración de tecnologías como HTTP, Express y Mongoose para gestionar y almacenar la información de manera eficiente.

Links

<https://github.com/luispl3105/Proyecto_ESP32_Node.js_Mongodb>

https://drive.google.com/drive/folders/1ssjt96V8lQ7HAQGUkkUl0LATTwsbwvtp?usp=drive\_link