Tecnológico de Monterrey

Manual técnico detallada del sistema

Implementación de Internet de las Cosas

Grupo 506 - TC 1004 B

Jose Manuel Sanchez Perez A01178230

Ricardo Gaspar Ochoa A00838841

Emiliano Enríquez López A01174554

Octavio Ramos Treviño A00840145

Edmundo Ruelas Angulo A01742824

20 de noviembre del 2024

Índice

Introducción

Este proyecto utiliza un sensor LDR (fotoresistor) para monitorear los niveles de luminosidad en un ambiente. Los datos registrados son enviados a un servidor para su almacenamiento en una base de datos y posteriormente analizados mediante gráficos en una interfaz frontend. El sistema incluye un microcontrolador ESP8266, un backend en Node.js, y un dashboard en Next.js.

El sistema proporciona métricas útiles como frecuencia de cambios de estado, tiempo promedio con luz baja, tiempo de LED activo, y análisis de operación por períodos de tiempo.

El valor registrado por el LDR estará en el rango de 0 a 1023:

- 0: Voltaje más bajo (oscuridad máxima).
- 1023: Voltaje más alto (luz máxima).

Hardware:

- ESP8266 con conexión Wi-Fi.
- Fotoresistor (LDR).
- Resistor de 10 kΩ.
- LED verde.

Software:

- **Microcontrolador**: Código en Arduino IDE.
- **Backend**: Servidor Node.js con conexión a MySQL.
- **Frontend**: Dashboard en Next.js con visualización en Chart.js.

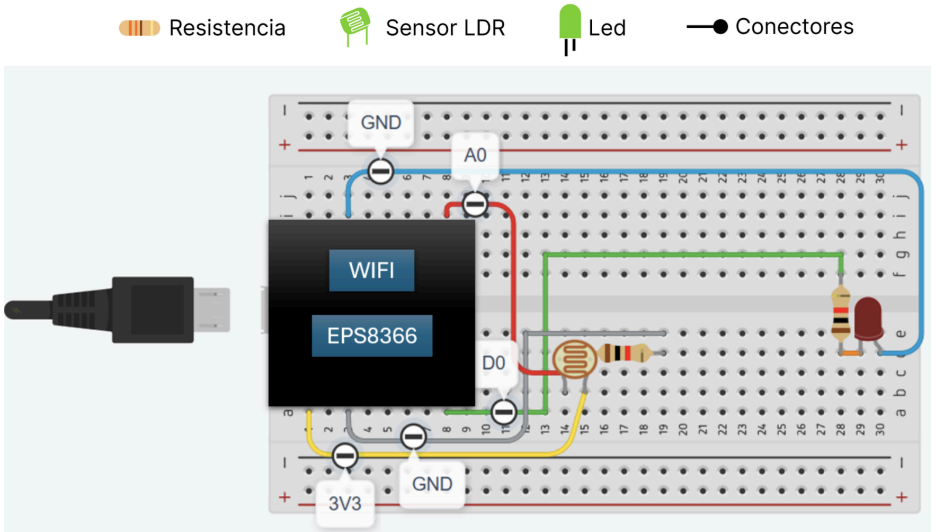


Imagen. Representación visual simplificada de los elementos de hardware.

Representación visual de los sistemas

El sistema empieza desde la conexión del circuito con el fotoresistor, el cual detecta los cambios de luz en el ambiente y transmite los valores al backend (el despliegue del backend está en la plataforma render.com o en local) y el Api, donde los procesa y registra en la base de datos (base de datos en la plataforma AIVEN o en local) . De esa misma forma podemos leer los datos de la base de datos y conectarlos a la plataforma web como Looker studio, donde podemos graficar y mostrar información de valor acorde a la data almacenada en la base de datos.

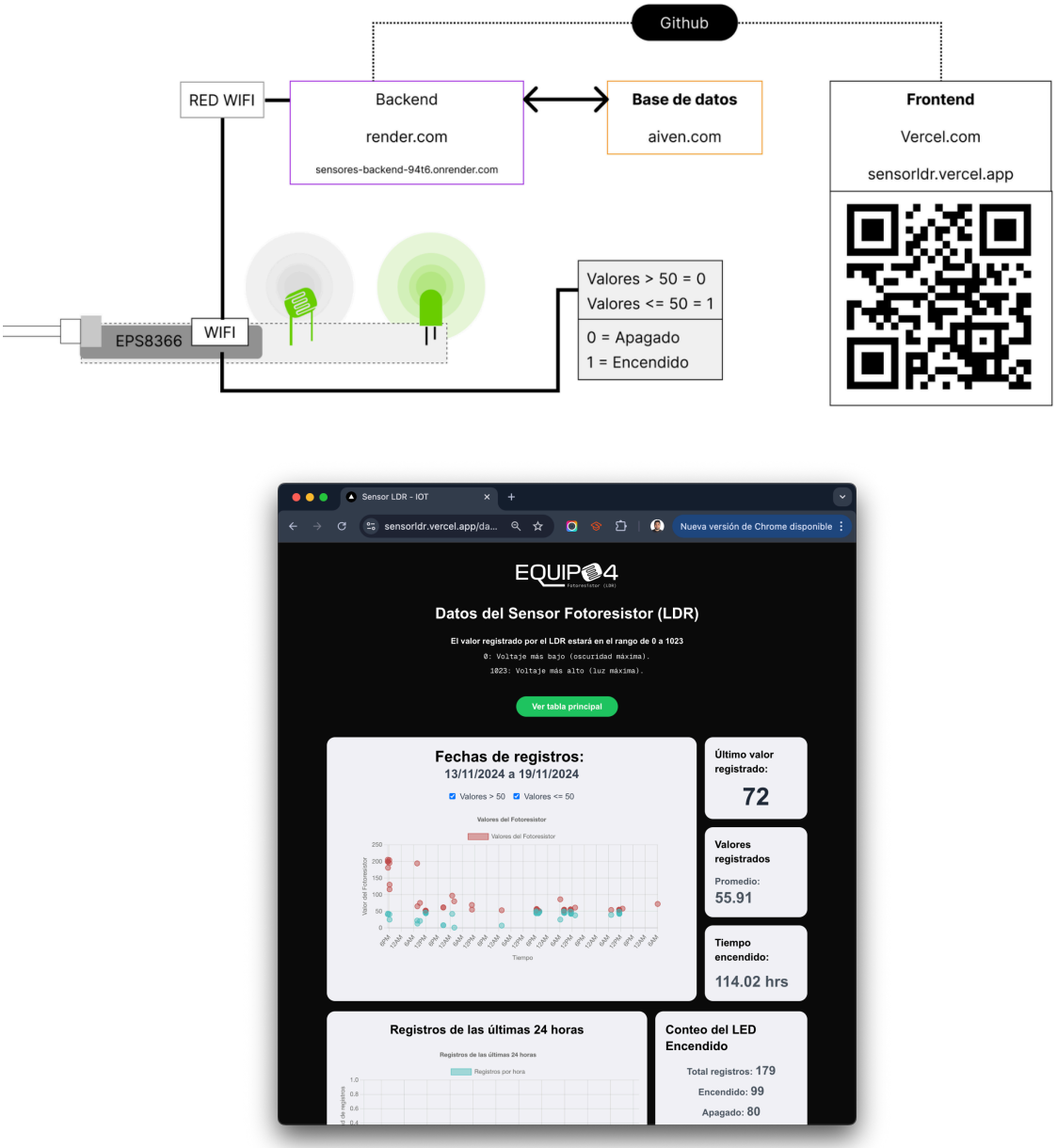


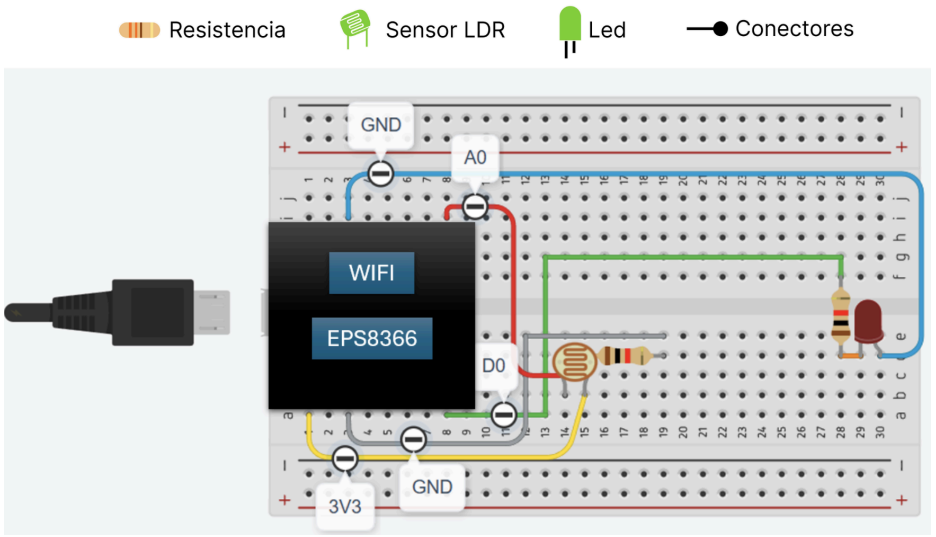
Imagen. Representación visual simplificada, visión general de un sistema final.

Arduino

Descripción de Funcionalidades

Microcontrolador (ESP8266)

- **Conexión Wi-Fi**: El ESP8266 se conecta a una red Wi-Fi especificada y envía datos al servidor backend mediante HTTPS.
- **Lectura del sensor LDR**: Se obtienen valores en un rango de 0 a 1023, donde valores altos indican mayor luminosidad.
- **Control del LED**: Si el valor del sensor está por debajo de 50, el LED se enciende indicando necesidad de iluminación artificial.
- **Envío de datos**: Los valores del LDR y el estado del LED son enviados al servidor backend en formato JSON.



Código Arduino IDE:

El código incluye:

- Configuración de red y servidor.
- Lectura de valores del LDR con un intervalo de 5 segundos.
- Control de estado del LED basado en el umbral de luminosidad.
- Envío de datos al backend utilizando HTTPS.

```javascript
#include <ESP8266WiFi.h>
#include <WiFiClientSecure.h>  // Usa WiFiClientSecure para HTTPS

const char* ssid = "";
const char* password = "";
```

```
const char* server = "";

WiFiClientSecure client;

int lastLDRValue = -1;
bool lastWasAbove50 = false;

void setup() {
 Serial.begin(9600);
 delay(10);
 pinMode(D0, OUTPUT);
  // Conexión a la red Wi-Fi
 Serial.println("Conectando a WiFi...");
 WiFi.begin(ssid, password);

 while (WiFi.status() != WL_CONNECTED) {
   delay(1000);
   Serial.print(".");
 }
 Serial.println("Conectado a WiFi");

 // Establecer una conexión segura (ignora la verificación de certificado)
 client.setInsecure();  // Opcional: Ignora la verificación del certificado
(no recomendado para producción)
}

void loop() {
 int ldr = analogRead(A0);
 Serial.print("Valor del LDR: ");
 Serial.println(ldr);

 int modo_operacion = (ldr > 50) ? 0 : 1;

 if ((ldr >= 50 && !lastWasAbove50) || (ldr < 50 && lastWasAbove50)) {
   lastLDRValue = ldr;
   lastWasAbove50 = (ldr >= 50);

   if (ldr >= 500) {
     digitalWrite(D0, LOW);
   } else if (ldr < 50) {
     digitalWrite(D0, HIGH);
   } else {
     digitalWrite(D0, LOW);
   }

   // Conectar al servidor HTTPS en el puerto 443
   if (client.connect(server, 443)) {
     Serial.println("Conectado al servidor");
```

```
      // Preparar los datos en formato JSON
      String postData = "{\"ldr\":" + String(ldr) + ", \"modo_operacion\":" +
String(modo_operacion) + "}";

      // Enviar la solicitud HTTP POST
      client.print(String("POST /sensor_datos HTTP/1.1\r\n") +
                   "Host: " + server + "\r\n" +
                   "Content-Type: application/json\r\n" +
                   "Content-Length: " + postData.length() + "\r\n" +
                   "Connection: close\r\n\r\n" +
                   postData);

      // Leer la respuesta del servidor
      String response = client.readString();
      Serial.println("Respuesta del servidor:");
      Serial.println(response);

      client.stop();
    } else {
      Serial.println("Conexión fallida al servidor");
    }
  }

  delay(5000);
}
```

Backend

Backend

- **Framework**: Node.js con Express.
- **Base de datos**: MySQL para almacenar registros del sensor.
- **Endpoints**:
    - `POST /sensor_datos`: Registra los datos del sensor en la base de datos.
    - `GET /sensor_datos`: Recupera los registros almacenados para análisis.

Código Backend:
Link al repositorio: [Link a Github]

- Configuración de conexión segura a la base de datos.
- Inserción de datos en formato JSON incluyendo la hora de registro.
- Recuperación de datos con ordenación por tiempo.

Frontend

Frontend

- **Framework**: Next.js.
- **Visualización de datos**: Utiliza Chart.js para gráficos y tablas interactivas.

Funcionalidades del Dashboard:

1. **Gráficos**:
    - **Bubble Chart**: Representa valores del fotoresistor en función del tiempo.
    - **Bar Chart**: Registros agrupados por hora en las últimas 24 horas.
    - **Pie Chart**: Proporción de tiempo con LED encendido y apagado.
2. **Indicadores**:
    - Promedio de valores del fotoresistor.
    - Tiempo total con LED encendido.
    - Último valor registrado.
3. **Interacción**:
    - Filtros para mostrar datos por rangos de valores.
    - Tabla detallada con todos los registros.

Código Frontend:
Link al repositorio: Link a Github

- Estructurado como un componente React.
- Incluye lógica para obtener y procesar datos desde el backend.
- Gráficos dinámicos con filtros aplicados en tiempo real.

Métricas generadas

1. **Frecuencia de Activación del Sensor**:
    - Cuántas veces el valor cambia entre luz baja/alta.
2. **Promedio de Valor del Fotoresistor**:
    - Cálculo del promedio en intervalos de tiempo específicos.
3. **Porcentaje de Tiempo con Luz Baja**:
    - Porcentaje del tiempo en que el sensor registra valores por debajo de 50.
4. **Porcentaje de Tiempo con el LED Activo**:
    - Relación entre el tiempo con LED encendido y el tiempo total.
5. **Registro de Tiempo en Estado Luminoso**:
    - Promedio de tiempo continuo con luz alta antes de un cambio a luz baja.
6. **Historial de Cambios de Estado**:
    - Registro detallado de cada transición de luz.
7. **Análisis de Operación por Hora/Día**:
    - Activación del sensor y LED por franjas horarias.

Configuración y despliegue

Microcontrolador:

1. Configurar la red Wi-Fi y servidor en las constantes `ssid`, `password`, y `server`.
2. Subir el código al ESP8266 desde Arduino IDE.
3. Conectar el LDR y el LED según el esquema de pines.

Backend:

1. Configurar variables de entorno (`DB_HOST`, `DB_USER`, etc.).
2. Instalar dependencias: `npm install`.
3. Iniciar servidor: `node index.js`.

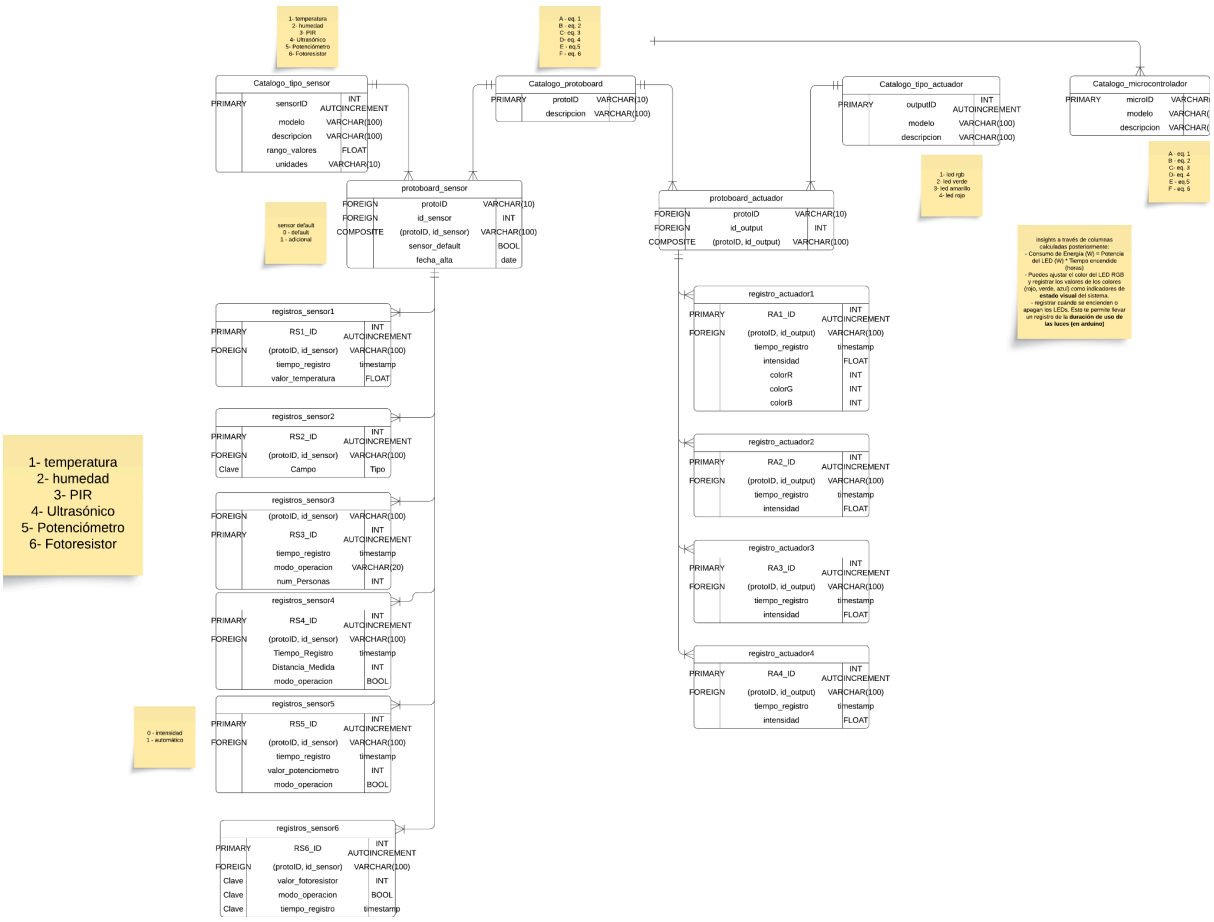
Frontend:

1. Configurar la URL del backend en el componente.
2. Iniciar el proyecto Next.js: `npm run dev`.
3. Acceder al dashboard en `http://localhost:3000`.

Base de datos

Diagramas de base de datos

Link a LucidApp : Link al diagrama.

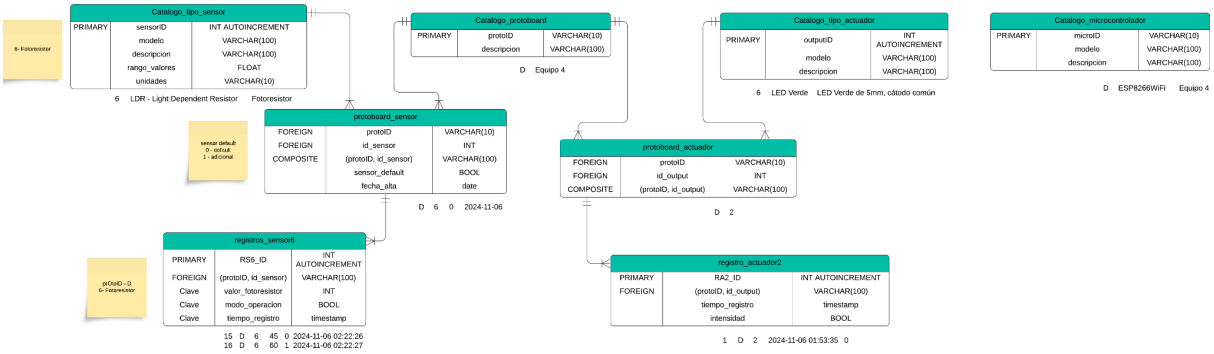


Tablas principales para los registros del equipos 4:

El diseño de la base de datos se estructura en tablas que organizan los diferentes elementos de sensores, protoboards, actuadores y microcontroladores, con sus respectivas relaciones y registros de operación. La tabla **`Catalogo_tipo_sensor`** define los sensores disponibles, incluyendo su modelo, descripción, rango de valores y unidades de medida. **`Catalogo_protoboard`** y **`Catalogo_tipo_actuador`** contienen las especificaciones de las protoboards y actuadores, respectivamente, cada uno con su modelo y descripción. **`Catalogo_microcontrolador`** almacena los microcontroladores con sus características principales.

Para gestionar las conexiones entre dispositivos, la tabla **`protoboard_sensor`** relaciona las protoboards con los sensores, indicando si un sensor es predeterminado y registrando la fecha de alta. De manera similar, **`protoboard_actuador`** enlaza la protoboard con los actuadores. Las tablas **`registros_sensor6`** y **`registro_actuador2`** registran las lecturas y estados de los sensores y actuadores, respectivamente. **`registros_sensor6`** almacena el valor capturado por el fotoresistor y el modo de operación del sensor, mientras que **`registro_actuador2`** guarda la intensidad del actuador y el tiempo de activación. Cada tabla incluye claves primarias y foráneas para asegurar la integridad referencial entre las entidades.

Link a LucidApp: [Link al diagrama](#).



Diccionario de Datos de la base de datos

Nombre de la tabla	Nombre del dato	Tipo de dato (dominio)	Descripcion
Catalogo_tipo_sensor	sensor_id	INT, PRIMARY KEY, AUTO_INCREMENT	Identificador único del sensor.
Catalogo_tipo_sensor	modelo	VARCHAR(100)	Modelo del sensor.
Catalogo_tipo_sensor	descripcion	VARCHAR(100)	Descripción breve del sensor.
Catalogo_tipo_sensor	rango_valores	FLOAT	Rango de valores que el sensor puede medir.
Catalogo_tipo_sensor	unidades	VARCHAR(10)	Unidad de medida del sensor.
Catalogo_protoboard	protoID	VARCHAR(10), PRIMARY KEY	Identificador único de la protoboard.
Catalogo_protoboard	descripcion	VARCHAR(100)	Descripción de la protoboard.
Catalogo_tipo_actuador	outputID	INT, PRIMARY KEY, AUTO_INCREMENT	Identificador único del actuador.
Catalogo_tipo_actuador	modelo	VARCHAR(100)	Modelo del actuador.
Catalogo_tipo_actuador	descripcion	VARCHAR(100)	Descripción breve del actuador.
Catalogo_microcontrolador	microID	VARCHAR(10), PRIMARY KEY	Identificador único del microcontrolador.

Catalogo_microcontrolador	modelo	VARCHAR(100)	Modelo del microcontrolador.
Catalogo_microcontrolador	descripcion	VARCHAR(100)	Descripción breve del microcontrolador.
protoboard_sensor	protoID	VARCHAR(10), FOREIGN KEY	Identificador de la protoboard, referencia a `Catalogo_protoboard(protoID)`.
protoboard_sensor	id_sensor	INT, FOREIGN KEY	Identificador del sensor, referencia a `Catalogo_tipo_sensor(sensorID)`.
protoboard_sensor	sensor_default	BOOL	Indicador de si el sensor es el predeterminado (0 = default, 1 = adicional).
protoboard_sensor	fecha_alta	DATE	Fecha de registro del sensor en la protoboard.
protoboard_actuador	protoID	VARCHAR(10), FOREIGN KEY	Identificador de la protoboard, referencia a `Catalogo_protoboard(protoID)`.
protoboard_actuador	id_output	INT, FOREIGN KEY	Identificador del actuador, referencia a `Catalogo_tipo_actuador(outputID)`.
registros_sensor6	RS6_ID	INT, PRIMARY KEY, AUTO_INCREMENT	Identificador único del registro.
registros_sensor6	protoID	VARCHAR(10), FOREIGN KEY	Identificador de la protoboard, referencia a `protoboard_sens`

			`or(protoID, id_sensor)`.
registros_sensor6	sensorID	INT, DEFAULT 6, NOT NULL	Identificador del sensor.
registros_sensor6	tiempo_registro	TIMESTAMP, DEFAULT CURRENT_TIMESTAMP	Fecha y hora de registro del valor del sensor.
registros_sensor6	valor_fotoresistor	INT	Valor registrado por el fotoresistor.
registros_sensor6	modo_operacion	BOOL	Modo de operación del sensor (0 = intensidad, 1 = cambio de color).
registro_actuador2	RA2_ID	INT, PRIMARY KEY, AUTO_INCREMENT	Identificador único del registro.
registro_actuador2	protoID	VARCHAR(10), FOREIGN KEY	Identificador de la protoboard, referencia a `protoboard_actuador(protoID, id_output)`.
registro_actuador2	outputID	INT, DEFAULT 2, NOT NULL	Identificador del actuador.
registro_actuador2	tiempo_registro	DATETIME, DEFAULT CURRENT_TIMESTAMP	Fecha y hora del registro del actuador.
registro_actuador2	intensidad	BOOL	Estado de intensidad del actuador (0 = apagado, 1 = prendido).

Código SQL

```
JavaScript
-- NO se deben agregar más datos
```

```sql
CREATE TABLE `Catalogo_tipo_sensor` (
  `sensorID` INT primary key AUTO_INCREMENT,
  `modelo` VARCHAR(100),
  `descripcion` VARCHAR(100),
  `rango_valores` FLOAT,
  `unidades` VARCHAR(10)
);

insert into Catalogo_tipo_sensor (modelo, descripcion) values
             ('LM35','Sensor de Temperatura'), -- id 1
             ('DHT11','Sensor de Humedad'), -- id 2
             ('HC-SR501','Sensor PIR'), -- -- id 3
             ('HC-SR04','Sensor Ultrasónico'), -- -- id 4
             ('10kΩ','Potenciometro'), -- id 5
             ('LDR - Light Dependent Resistor','Fotoresistor'); -- id 6



-- NO se deben agregar más datos
CREATE TABLE `Catalogo_protoboard` (
  `protoID` VARCHAR(10) primary key,
  `descripcion` VARCHAR(100)
);

insert into Catalogo_protoboard (protoID, descripcion) values
             ('A','Equipo 1'),
             ('B','Equipo 2'),
             ('C','Equipo 3'),
             ('D','Equipo 4'),
             ('E','Equipo 5'),
             ('F','Equipo 6');



-- NO se deben agregar más datos
CREATE TABLE `Catalogo_tipo_actuador` (
  `outputID` INT primary key AUTO_INCREMENT,
  `modelo` VARCHAR(100),
  `descripcion` VARCHAR(100)
);

insert into Catalogo_tipo_actuador (modelo, descripcion) values
             ('LED RGB 4 patas','LED RGB de cátodo común'), -- id 1
             ('LED Verde','LED Verde de 5mm, cátodo común'), -- id 2
             ('LED Amarillo','LED Amarillo de 5mm, cátodo común'), -- id 3
             ('LED Rojo','LED Rojo de 5mm, cátodo común'); -- id 4
```

```sql
-- NO se deben agregar más datos
CREATE TABLE `protoboard_sensor` (
  `protoID` VARCHAR(10),
  `sensorID` INT,
  `sensor_default` BOOL, -- ajuste: 0 falso, 1 verdadero. siempre 1 pues solo
usamos potenciómetro.
  `fecha_alta` date default curdate(),
  PRIMARY KEY (`protoID`, `sensorID`),
  FOREIGN KEY (`protoID`) REFERENCES `Catalogo_protoboard`(`protoID`),
  FOREIGN KEY (`sensorID`) REFERENCES `Catalogo_tipo_sensor`(`sensorID`)
);

INSERT INTO `protoboard_sensor` (`protoID`, `sensorID`,`sensor_default`)
VALUES
('D', 6, 0);


-- Se agregan datos desde el BACKEND
CREATE TABLE `registros_sensor6` (
  `RS6_ID` INT primary key AUTO_INCREMENT,
  `protoID` VARCHAR(10) default 'D' not null,
  `sensorID` INT default 5 not null, -- id del sensor
  `tiempo_registro` TIMESTAMP default current_timestamp,
  `valor_fotoresistor` INT,
  `modo_operacion` BOOL, -- 0 intensidad, 1 cambio de color
  FOREIGN KEY (`protoID`, `sensorID`) REFERENCES
`protoboard_sensor`(`protoID`, `sensorID`)
);

-- datos prueba
INSERT INTO `registros_sensor6` (`tiempo_registro`,`valor_fotoresistor`,
`modo_operacion`) VALUES
(45,0),
(60,1);


-- NO se deben agregar más datos
CREATE TABLE `protoboard_actuador` (
  `protoID` VARCHAR(10),
  `outputID` INT,
  PRIMARY KEY (`protoID`, `outputID`),
  FOREIGN KEY (`protoID`) REFERENCES `Catalogo_protoboard`(`protoID`),
  FOREIGN KEY (`outputID`) REFERENCES `Catalogo_tipo_actuador`(`outputID`)
);

INSERT INTO `protoboard_actuador` (`protoID`, `outputID`) VALUES
('D', 2);
```

```
-- Se agregan datos desde el BACKEND
CREATE TABLE `registro_actuador2` (
  `RA2_ID` INT AUTO_INCREMENT,
  `protoID` VARCHAR(10) default 'D' not NULL, -- default a id A (equipo 4)
  `outputID` INT default 2 not NULL, -- id del FOCO! default a id 2 (led
verde)
  `tiempo_registro` DATETIME default CURRENT_TIMESTAMP,
  `intensidad` BOOL, -- 0 apagado, 1 prendido,
  PRIMARY KEY (`RA2_ID`),
  FOREIGN KEY (`protoID`, `outputID`) REFERENCES
`protoboard_actuador`(`protoID`, `outputID`)
);

-- datos prueba
INSERT INTO `registro_actuador1` (`intensidad`) VALUES
(0);


CREATE TABLE Catalogo_microcontrolador (
    microID VARCHAR(10) PRIMARY KEY,
    modelo VARCHAR(100),
    descripcion VARCHAR(100)
);

INSERT INTO Catalogo_microcontrolador (microID, modelo, descripcion)
VALUES ('D', 'ESP8266WiFi', 'Equipo 4');
```

Conclusión

El sistema permite un monitoreo continuo de la luminosidad, ofreciendo análisis detallados para optimizar el uso de iluminación artificial. La integración entre hardware, backend y frontend proporciona una solución escalable y eficiente para análisis lumínico en tiempo real.

Referencias

Turbo Código. (2020, March 29). *Tutorial NodeMCU 01. Configura el ESP8266 y Primer programa* [Video]. YouTube. https://www.youtube.com/watch?v=YjqCQ0WNgyI