UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS
Universidad del Perú. Decana de América

FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
E.A.P. Ingeniería de Sistemas

Grupo: 4

Integrantes:

Guillen Piña, Ritcy Christina

Huaman Ramirez, Ana Cristina

Manrique Perez, Renzo Jheus

Villanueva Monrroy, Xavier Angel

Villavicencio de la Serna, Jimmy Manyear

Curso: Internet of Things

Sección: 1

Profesor: Rosas Cueva, Yessica

Lima, Perú

2023

Índice

1. Introducción

1.1. Revisión del estado del arte

Conforme crece el volumen de vehículos, nos vamos enfrentando a una crisis de la calidad del aire que cada vez se hace más acuciante en las grandes urbes. Tenemos como ejemplo la ciudad de Delhi, que, en el año 2021, era la ciudad más contaminada del mundo en términos de calidad del aire, superando incluso a Beijing, China.

Esta ciudad ha llegado a un nivel de partículas finas y contaminantes extremadamente peligrosos, a tal punto de que las autoridades tuvieron que tomar medidas extremas como cerrar escuelas y limitar la circulación de vehículos en las carreteras. Asimismo, según la BBC, esto ha sido causa directa de diferentes problemas de salud como enfermedades respiratorias, cardiovasculares y cáncer.

Sin irnos más lejos, en nuestro continente, según la NASA, los incendios masivos que se están volviendo más frecuentes en la Amazonía liberan alrededor de 228 megatoneladas de dióxido de carbono, lo que es equivalente a las emisiones anuales de dióxido de carbono de un país como España. A ello, la deforestación que se produce en la región también contribuye a la mala calidad del aire, ya que reduce la capacidad de los árboles para absorber dióxido de carbono y producir oxígeno.

En ese contexto, se ha vuelto de crucial importancia, tomar medidas de prevención para ser capaces de estimar cuando nuestro hogar supera el umbral “seguro” y la calidad del aire empieza a ser perjudicial para la vida y potencial fuente de enfermedades.

Con respecto al tema, existen múltiples artículos que han usado IoT para tratar de dar soluciones a este problema. Por lo que este proyecto no trata de crear algo nuevo, sino

indagar a detalle en dichas soluciones y crear alternativas más baratas y fáciles de implementar.

En primer lugar, se tiene la investigación de Wenhao, Yuan, Yanju, Jianbo y Weiwei. Ellos en su artículo de revisión examinan la tecnología y las aplicaciones de los sistemas de monitoreo de calidad del aire en interiores, y cómo pueden mejorar la salud y el bienestar de las personas. Los autores discuten los diferentes tipos de sensores y tecnologías de monitoreo disponibles, y cómo pueden detectar contaminantes del aire, como el monóxido de carbono y los compuestos orgánicos volátiles. También se discuten los desafíos técnicos y de diseño que enfrentan los sistemas de monitoreo de calidad del aire, como la calibración y la integración con otros sistemas de edificios. Finalmente, se discuten las perspectivas futuras para el desarrollo y la aplicación de sistemas de monitoreo de calidad del aire en interiores, incluyendo el uso de inteligencia artificial y el análisis de Big Data para mejorar la precisión y la eficiencia del monitoreo. Al respecto de este artículo, se destaca de que ellos mencionan que aún hay mucho espacio de mejorar para mejorar la eficacia de estos dispositivos.

En segundo lugar, en un plano más práctico, los autores Kaushik, Manjunath y Geetha describen un sistema de monitoreo inteligente de calidad del aire en interiores que utiliza sensores de gas y partículas para detectar la presencia de contaminantes en el aire. El sistema también utiliza un modelo de aprendizaje automático para predecir el riesgo de síndrome de edificio enfermo y proporcionar una alerta temprana a los ocupantes del edificio. Los autores describen los detalles del diseño y la implementación del sistema, así como los resultados de las pruebas en un edificio de oficinas en Bangalore, India. En cuanto a los sensores usados en este proyecto, estos estaban configurados para detectar la presencia de contaminantes en el aire, incluyendo dióxido de carbono, monóxido de carbono, dióxido de nitrógeno, ozono y partículas finas. En cuanto a la duración del experimento, este abordo un período de seis

meses, y se alcanzó un logro significativo, dado que el sistema logró el objetivo planteado. Sin embargo, aún existen limitaciones como la necesidad de calibración regular y la dificultad de integrar el sistema con otros sistemas de edificios.

Finalmente, también buscamos artículos orientados al desarrollo del sistema mencionado pero que use componentes de bajo coste. En este rubro, nos resultó útil el artículo de Yeong-Ren, Tse-Hao y Jun-Nan Lin. En dicho artículo, se describe un sistema de monitoreo de calidad del aire en interiores de bajo costo que utiliza un sensor de partículas para detectar la presencia de partículas finas PM2.5 y PM10 en el aire. El sistema también incluye un microcontrolador y una conexión inalámbrica para enviar datos a un servidor en la nube para su posterior análisis. Los autores describen los detalles del diseño y la implementación del sistema, así como los resultados de las pruebas en un ambiente cerrado y una comparación con un monitor comercial. En este caso, el artículo no llega a dar detalle acerca de las herramientas empleadas para la creación del servidor en la nube ni las herramientas empleadas para recopilar datos. Sin embargo, se destaca que en este proyecto se usa un microcontrolador ESP8266 debido a su bajo coste y bajo consumo de energía. Asimismo, debido a su capacidad de conexión inalámbrica a la red Wi-Fi, este dispositivo es ideal para un sistema económico pero eficaz. En el caso de este proyecto, se uso un Arduino UNO debido a la más amplia documentación existente para configurarlo con los sensores del proyecto. Sin embargo, también consideramos el ESP32 en planteamientos anteriores.

1.2. Planteamiento del problema

En este proyecto, nos proponemos abordar el problema de la mala calidad del aire en interiores, que puede tener un impacto negativo en la salud y el bienestar de las personas. A menudo, las personas pasan la mayor parte de su tiempo en interiores, ya sea en casa o en la oficina, lo que significa que están expuestas a la contaminación del aire interior durante

largos períodos de tiempo. Esto puede aumentar el riesgo de enfermedades respiratorias, especialmente en personas con asma, alergias o enfermedades crónicas.

Para abordar este problema, nuestro proyecto consiste en desarrollar un dispositivo de monitoreo de calidad del aire en interiores de bajo costo que pueda detectar la presencia de gases tóxicos, como el monóxido de carbono, y ajustar la temperatura y la humedad para mantener un ambiente cómodo y saludable. Creemos que nuestro dispositivo puede ayudar a mejorar la calidad de vida de las personas y reducir el riesgo de enfermedades respiratorias al proporcionar información en tiempo real sobre la calidad del aire interior y al permitir ajustar el ambiente de manera apropiada.

1.3. Objetivos

Nuestros objetivos para este proyecto, alineados con los objetivos de desarrollo sostenible, son los siguientes:

- Desarrollar un dispositivo de monitoreo de calidad del aire en interiores de bajo costo que pueda detectar la presencia de gases tóxicos, como el monóxido de carbono, y ajustar la temperatura y la humedad para mantener un ambiente cómodo y saludable. Asimismo, como parte de un plan de seguridad integral, se busca agregar una alarma de seguridad para la puerta.
- Realizar pruebas exhaustivas del dispositivo en Node-Red para asegurarnos de que sea confiable y preciso en la detección de gases tóxicos y en el monitoreo de la calidad del aire en interiores.
- Proporcionar una interfaz de usuario fácil de usar en Node-Red que permita a los usuarios ver los datos en tiempo real sobre la calidad del aire en interiores y hacer ajustes según sea necesario.

- Identificar posibles mejoras en el dispositivo y en su capacidad de monitorear y ajustar la calidad del aire en interiores, y explorar la posibilidad de integrar nuevas tecnologías en futuras versiones del dispositivo.

2. Marco Teórico

2.1. Proteus versión 8.13

Proteus 8.13 es un software de diseño electrónico que permite la creación de esquemas y simulaciones de circuitos. Ofrece una amplia variedad de herramientas y funciones, incluyendo la simulación de microcontroladores, la creación de componentes personalizados y la integración de esquemas y PCB. También cuenta con una interfaz de usuario fácil de usar para usuarios de todos los niveles de experiencia en diseño electrónico.

2.2. Librería de Arduino para Proteus

La librería de Arduino para Proteus es una colección de componentes y módulos electrónicos que se utilizan para simular proyectos basados en la plataforma de desarrollo Arduino en el software de diseño electrónico Proteus. Con esta librería, los diseñadores pueden simular diferentes componentes de hardware de proyectos de Arduino, incluyendo sensores, módulos de comunicación y actuadores, entre otros. La librería permite a los usuarios probar el comportamiento de sus proyectos de Arduino en Proteus antes de construirlos en el mundo real, lo que reduce el riesgo de errores costosos y acelera el proceso de desarrollo.

2.3. Node-Red

Node-RED es una herramienta basada en la web utilizada para crear flujos de trabajo automatizados para dispositivos y servicios de IoT. Utiliza un editor gráfico para conectar

nodos que representan dispositivos, servicios y acciones, lo que facilita la integración de diferentes sistemas y dispositivos. Node-RED tiene una variedad de paquetes adicionales que permiten la integración con diferentes servicios y protocolos de comunicación de IoT, lo que aumenta su flexibilidad y capacidades. Node-RED es de código abierto y tiene una comunidad activa de usuarios que contribuyen con nuevos nodos y recursos para la plataforma, lo que la convierte en una herramienta popular en el campo de IoT.

2.4. Virtual Serial Port Driver

El Virtual Serial Port Driver es un software que permite la creación de pares de puertos serie virtuales en una computadora. Estos pares de puertos constan de dos puertos virtuales, uno llamado "puerto COM virtual" y otro llamado "puerto COM virtual emparejado", que se comunican entre sí. Este software es útil para emular la comunicación serial entre dispositivos o programas en una computadora, sin necesidad de hardware adicional. Por ejemplo, se puede utilizar para conectar aplicaciones o programas que requieren una comunicación serial con un dispositivo real, a través de un puerto serial virtual en la computadora.

3. Componentes del Sistema

A continuación, se describirán los diversos componentes que se usaron en Proteus para el presente proyecto:

3.1. Arduino UNO

El Arduino UNO es una placa que cuenta con una serie de pines digitales y analógicos que pueden ser programados para realizar diferentes tareas, como leer entradas de sensores,

controlar motores y luces, y enviar señales a otros dispositivos. Además, incluye una interfaz USB que permite la comunicación con un ordenador para cargar programas y recibir datos.

En Proteus, se puede simular el Arduino UNO junto con otros componentes electrónicos para crear prototipos y probar el funcionamiento de proyectos sin necesidad de construirlos físicamente.

3.2. Gas Sensor MQ2

En Proteus, el sensor de gas utilizado para simular una alarma de humo es el "Gas Sensor MQ2". Este sensor es capaz de detectar varios gases combustibles, como el butano, el propano y el metano, así como el humo y otros gases tóxicos.

El Gas Sensor MQ2 funciona mediante la medición de la conductividad de un material sensible al gas. Cuando los gases inflamables o tóxicos están presentes, la conductividad del material cambia, lo que se refleja en un cambio en la resistencia eléctrica. El sensor envía esta información al microcontrolador a través de una señal analógica.

En la simulación de Proteus, el Gas Sensor MQ2 se conecta a un microcontrolador, como Arduino o PIC, que se encarga de procesar la información recibida del sensor y activar una alarma de humo en caso de detectar una concentración peligrosa de gases en el aire.

3.3. DHT11

En Proteus, el sensor de temperatura y humedad utilizado para simular un ambiente cerrado es el "DHT11". Este sensor es capaz de medir la temperatura y la humedad relativa del aire en un rango de 0 a 50 grados Celsius y 20% a 90% de humedad relativa.

El DHT11 funciona mediante la medición de la resistencia eléctrica de un material sensible a la humedad. Cuando la humedad del aire cambia, la resistencia eléctrica también

cambia, lo que se refleja en una señal analógica que se envía al microcontrolador. La temperatura se mide a través de un termistor que también produce una señal analógica en función de la temperatura.

En la simulación de Proteus, el sensor DHT11 se conecta a un microcontrolador, como Arduino o PIC, que se encarga de procesar la información recibida del sensor y ajustar la temperatura y la humedad del ambiente cerrado para mantener un ambiente cómodo y saludable. El microcontrolador puede controlar dispositivos como un aire acondicionado o un humidificador para lograr el objetivo de mantener una temperatura y humedad adecuadas en el ambiente cerrado.

3.4. Sensor ultrasónico

El sensor ultrasónico utilizado en Proteus es un sensor de distancia que utiliza ondas de sonido para medir la distancia a un objeto. Este sensor se basa en la tecnología de tiempo de vuelo, en la que se mide el tiempo que tarda una señal de ultrasonido en rebotar en un objeto y regresar al sensor. El sensor emite una señal de ultrasonido que viaja a través del aire y, cuando se encuentra con un objeto, parte de la señal es reflejada de vuelta hacia el sensor. El tiempo que tarda la señal en regresar al sensor se utiliza para calcular la distancia al objeto. Este tipo de sensor se utiliza comúnmente en robots y sistemas de automatización para detectar objetos y medir distancias.

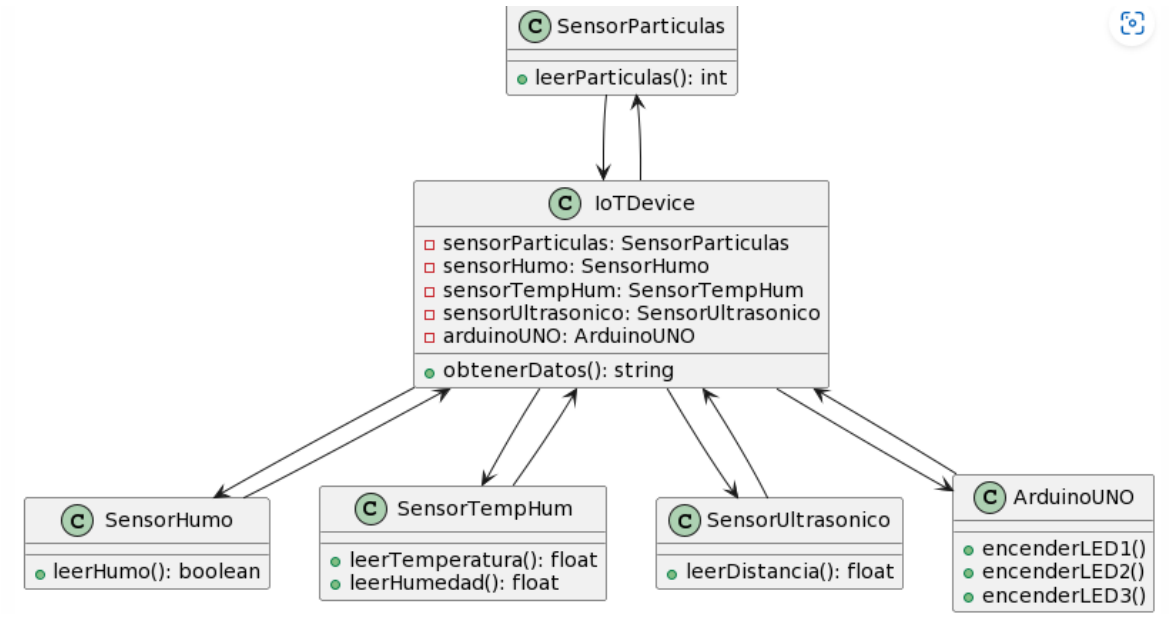
3.5. Sensor de partículas finas

El sensor de partículas finas utilizado en Proteus es un sensor que se utiliza para medir la cantidad de partículas finas en el aire. Este tipo de sensor se basa en la tecnología de dispersión láser, en la que un haz de luz láser es utilizado para iluminar las partículas en el aire. La luz se dispersa cuando se encuentra con las partículas, y la cantidad de luz dispersada

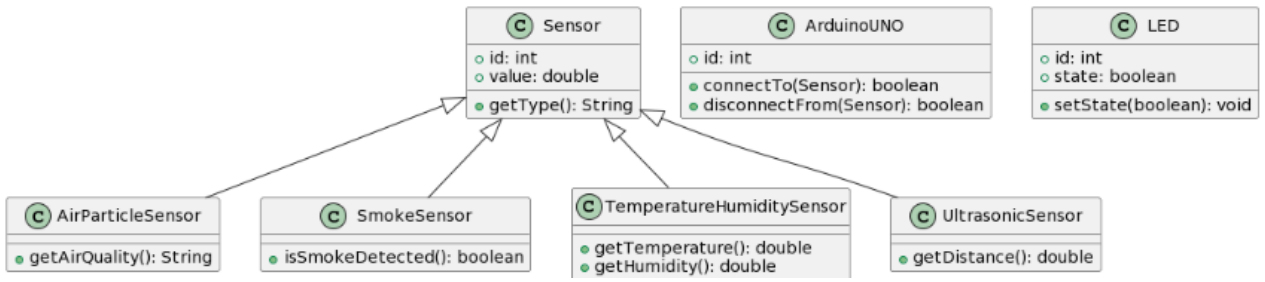
es medida por el sensor. A partir de la cantidad de luz dispersada, el sensor puede determinar la cantidad de partículas finas presentes en el aire y su tamaño.

4. Implementación del Sistema

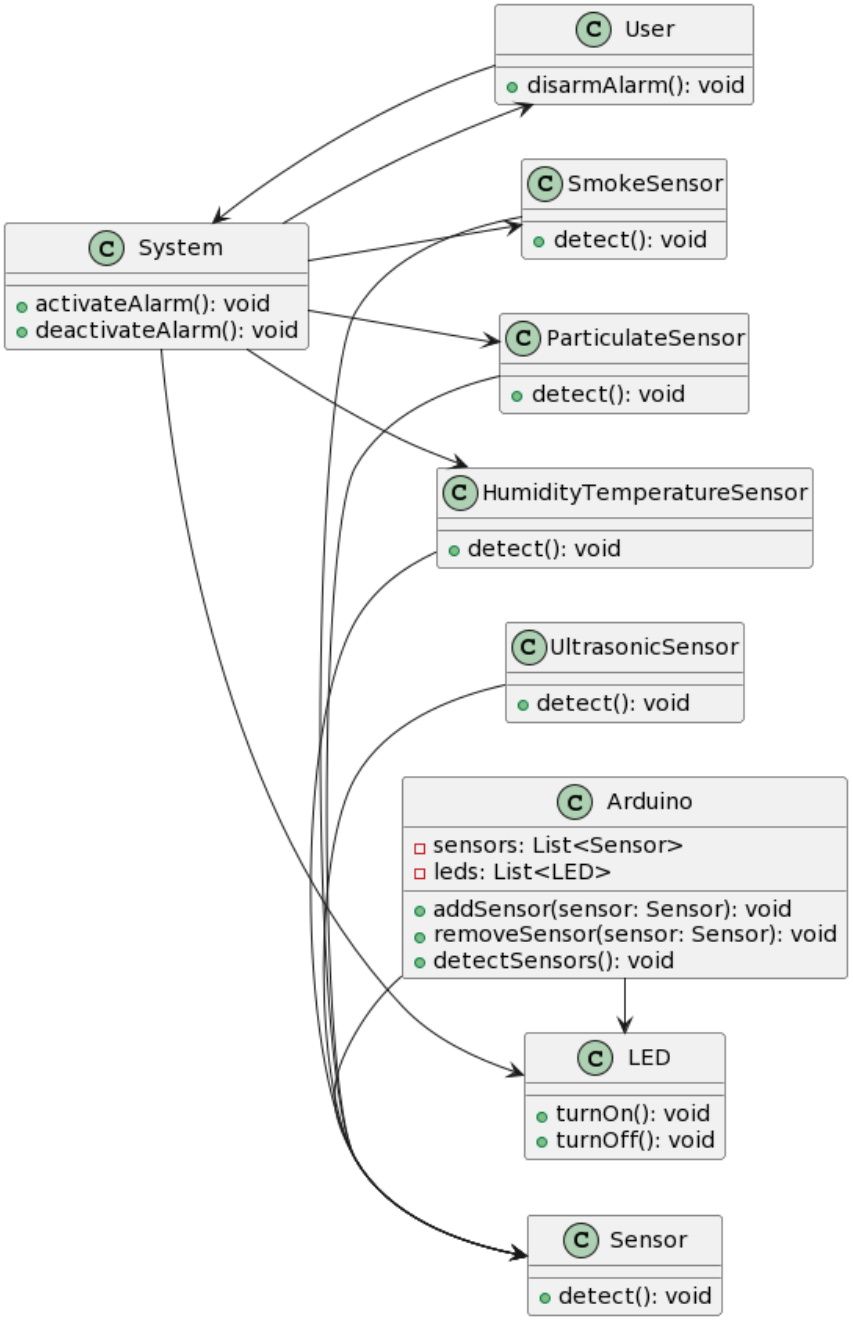
4.1. Diagrama de modelado



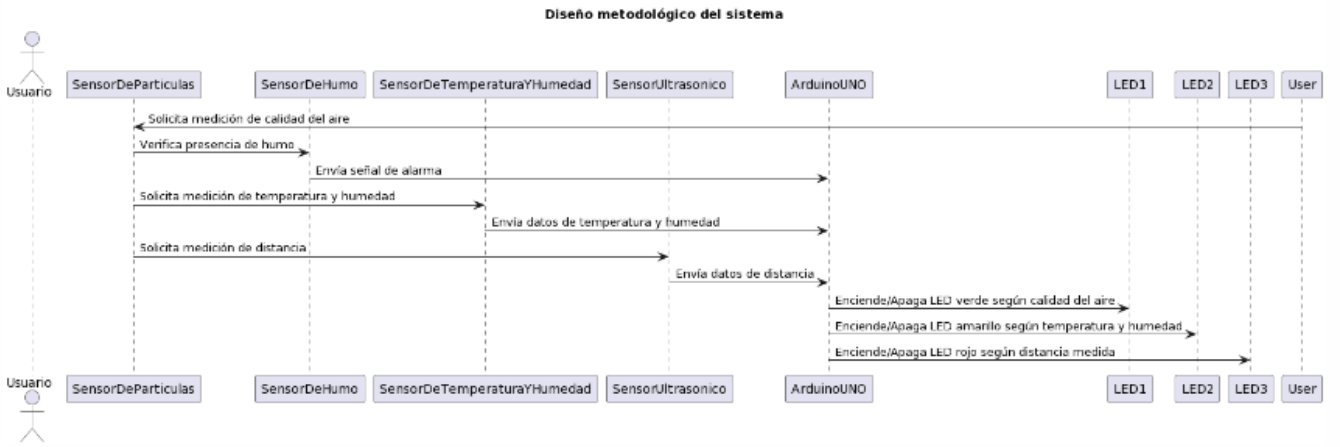
4.2. Diagrama de clases



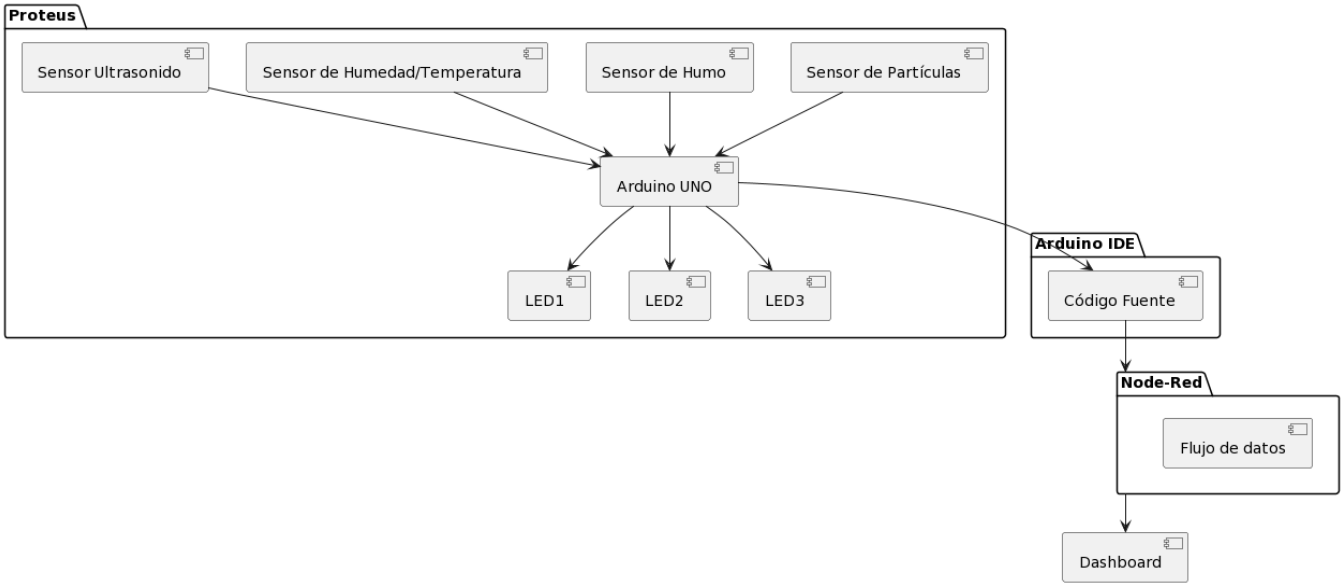
4.3. Diagrama de clases de uso



4.4. Diseño metodológico

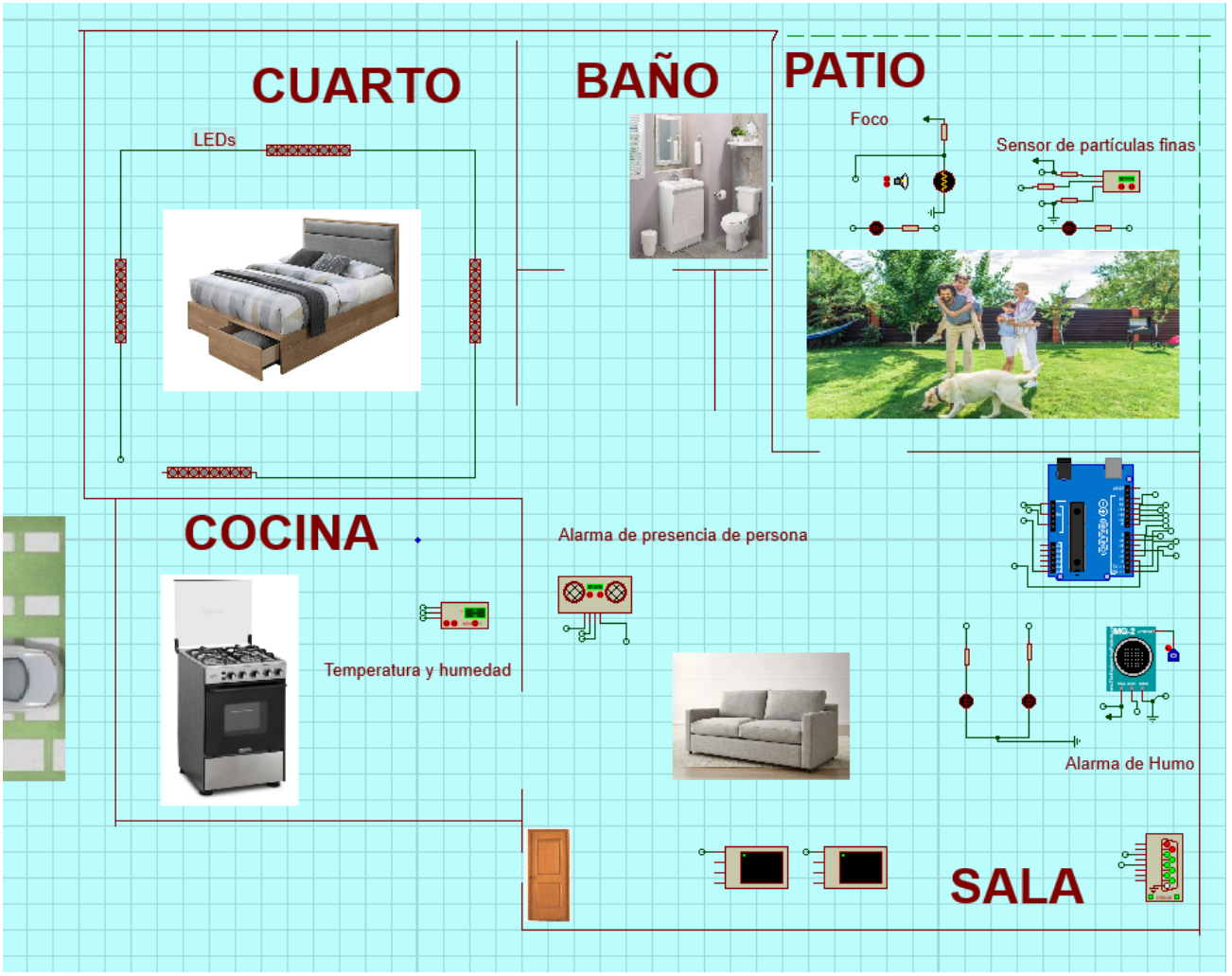


4.5. Diagrama de componentes



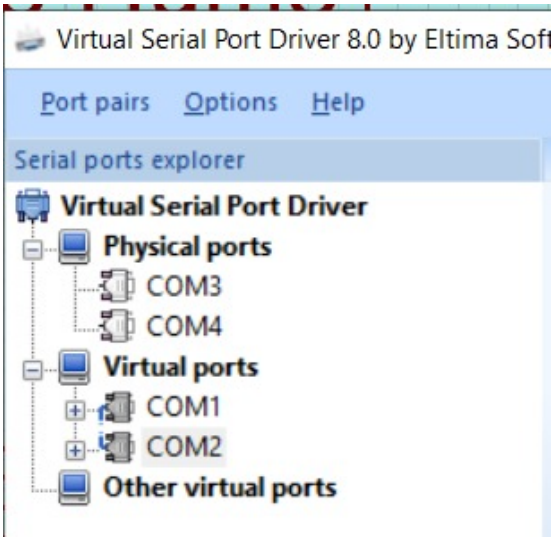
4.6. Simulación en Proteus

El circuito trabajado en Proteus fue el siguiente:

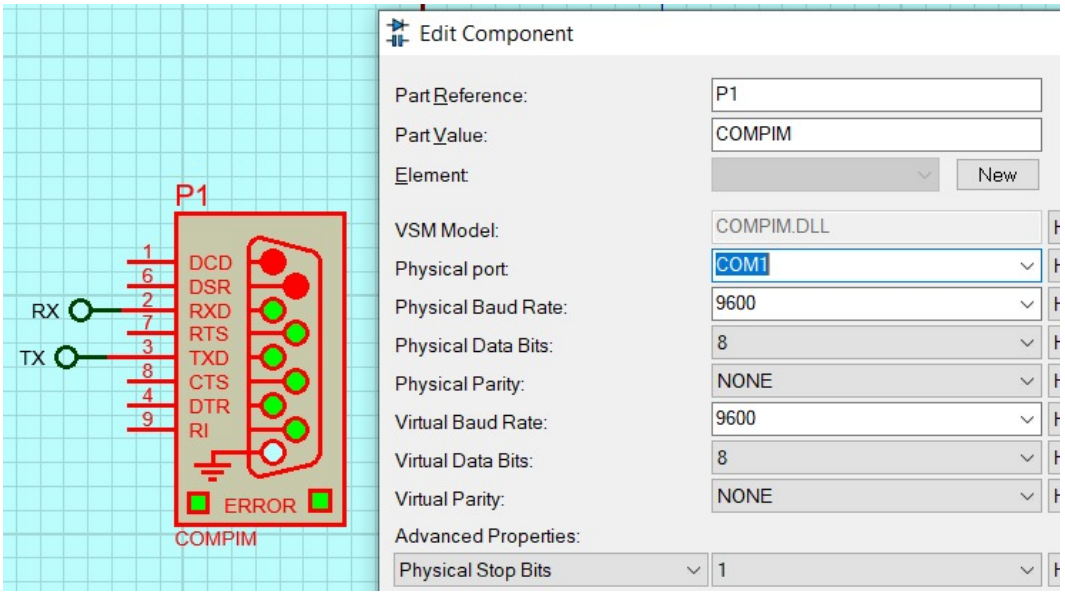


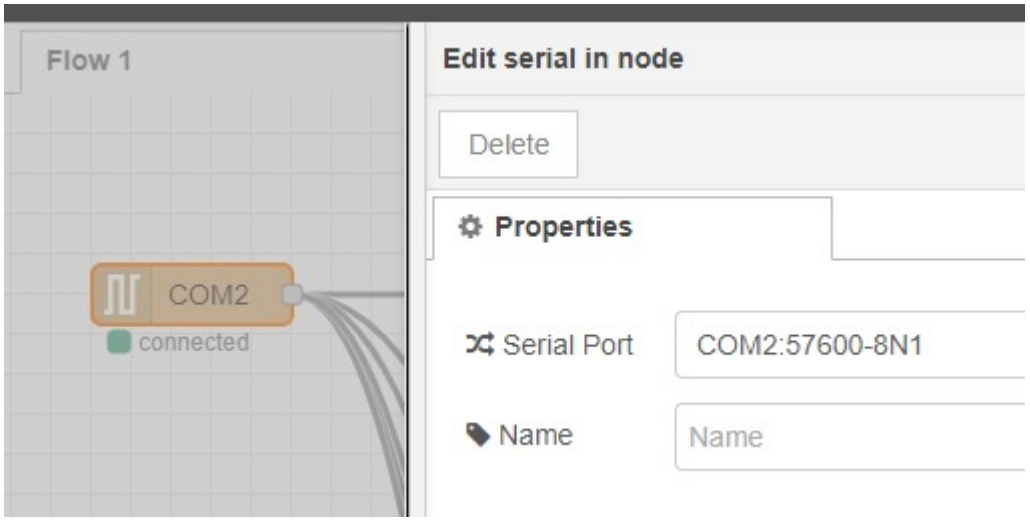
4.7. Node-Red

Para conectar el Node-Red lo podemos hacer de dos formas, una mediante señal wifi (Mqtt) y otro mediante un puertos seriales virtuales. Para este caso hemos elegido el segundo, por lo que primero que debemos hacer es descargar “Virtual Serial Port Driver” y crear dos puertos virtuales.

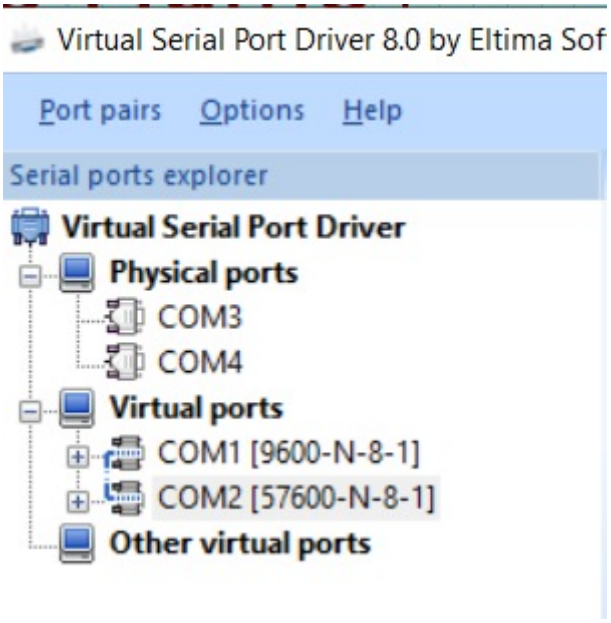


Ahora vamos a conectar el puerto COM1 a COMPIM en Proteus (de entrada) y el puerto COM2 al serial inicial de Node-Red (de salida).





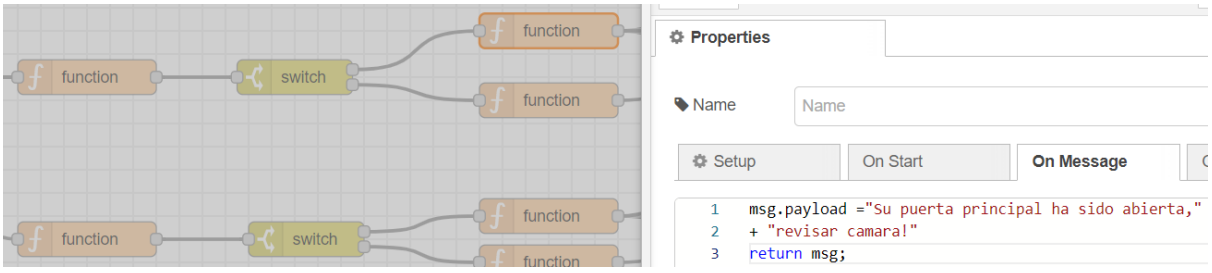
Ahora ya tenemos conectado Proteus a Node-Red, teniendo como resultado:



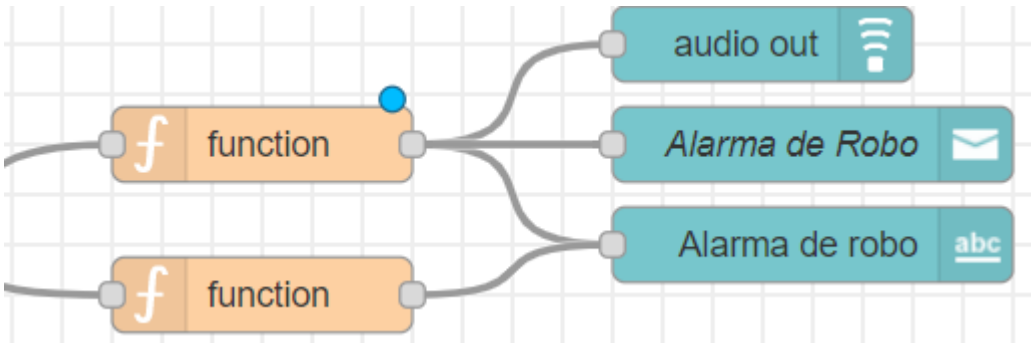
Como ya tenemos la conexión ahora si podemos proseguir a armar los nodos de Node-Red. En base a la cantidad de sensores, ponemos la cantidad de funciones.Tenemos que tomar en cuenta que los valores recomendados a mandar son 0 y 1 pues así podremos conseguir un código más amigable.

```
var puerta = msg.payload.split(',')[1];
msg.payload = puerta;
if(puerta == 1){
    msg.payload = true
}
else{
    msg.payload = false
}
return msg;
```

Nota: Primera función (sensor de proximidad) de muestra donde se utiliza el primer valor del arreglo utilizando una separación de “,” para poder continuar al siguiente mensaje.

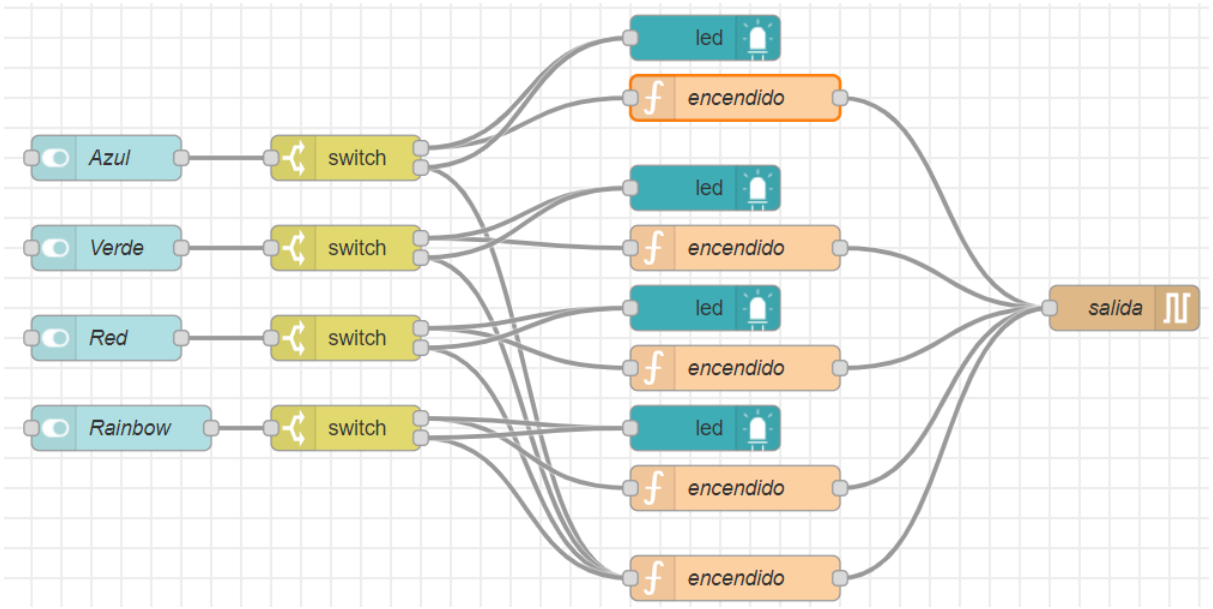


Nota: Paso siguiente a la función (sensor de proximidad) donde en caso de ser “true”, el mensaje que se mostrará será el mostrado en la función siguiente.

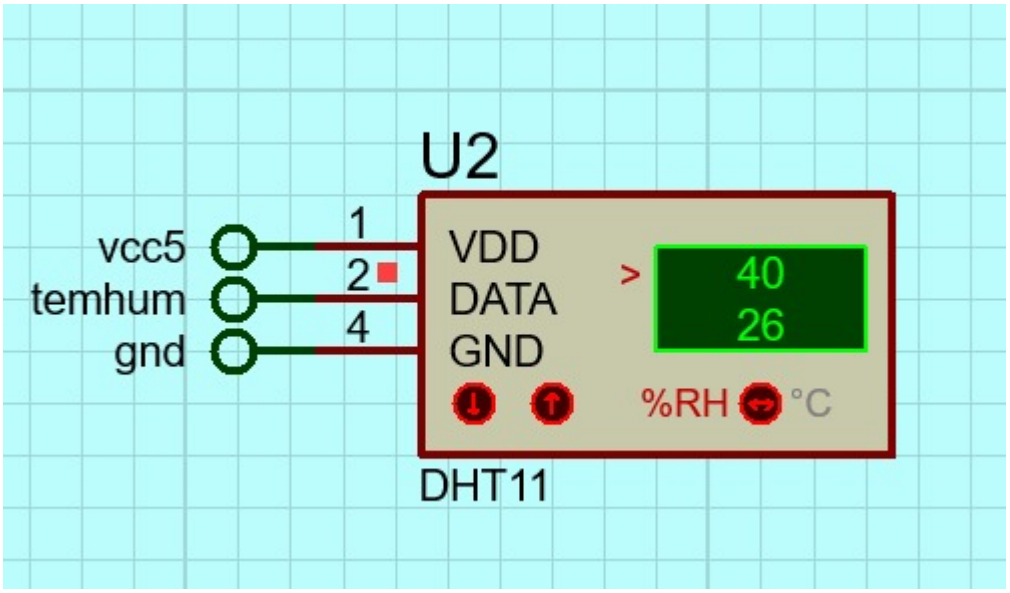


Nota: Formas de enviar el mensaje, en caso se retorna el valor “true”, de la función (sensor de proximidad) desde audio, mensaje al teléfono o alarma parpadeante.

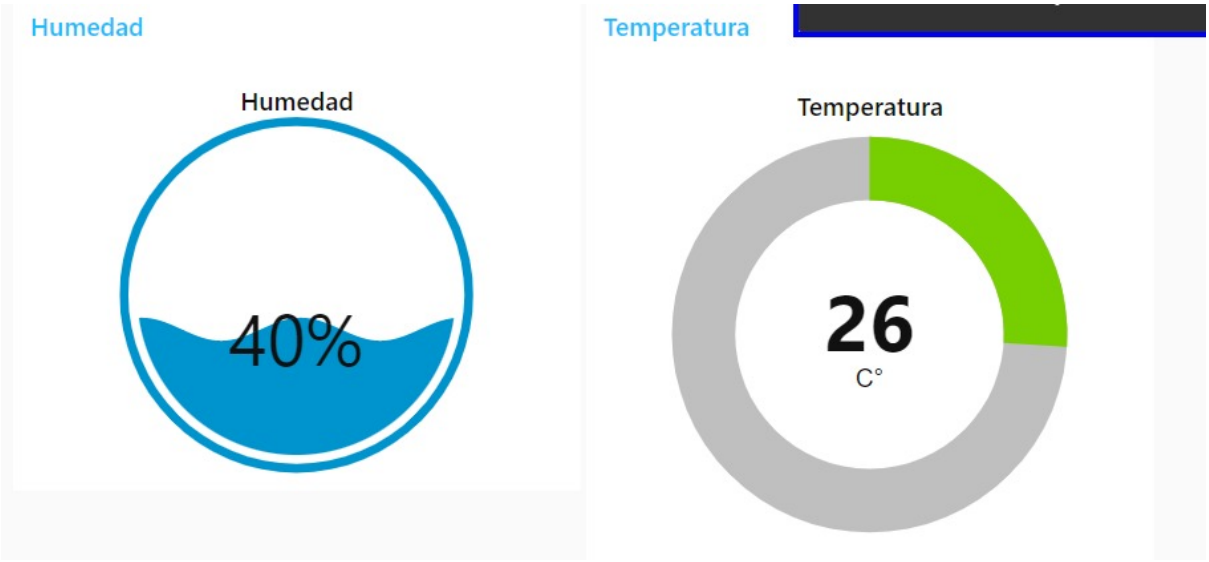
Los leds son configurados mediante el color que fueron programados en arduino, en cada uno dependiendo si son encendidos o apagados se mostrará un tipo de color en el led y en los focos del cuarto y patio.



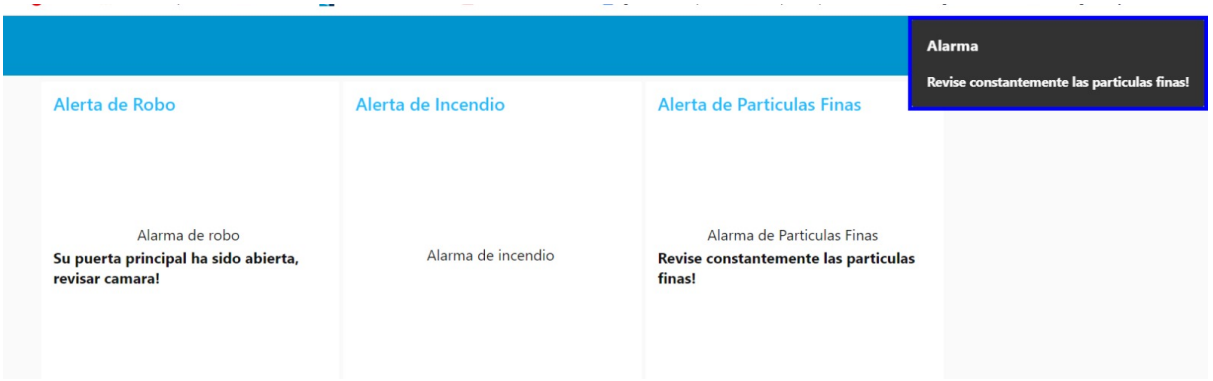
Luego de realizar la conexión de los nodos, en el dashboard podemos visualizar los datos que se nos son enviados por parte del Proteus.



Nota: Sensor de temperatura y humedad en Proteus

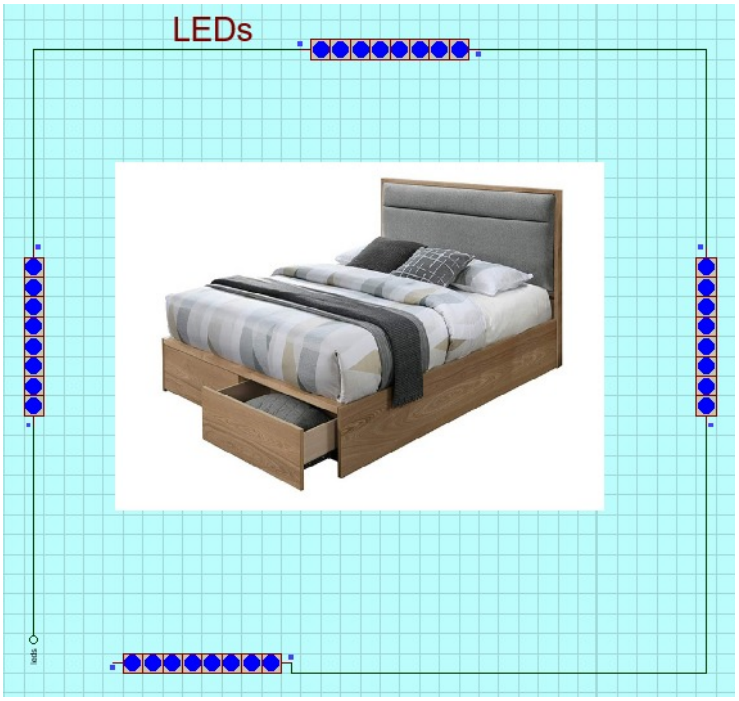


Nota: Valores de temperatura y húmedad recibidos del Proteus mostrados en el Node Dashboard

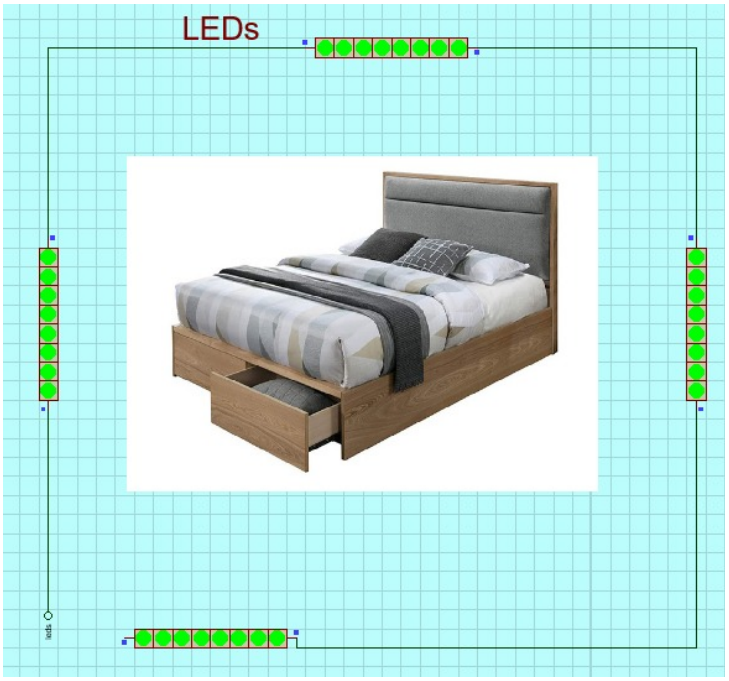


Nota: Valores de las alarmas de robo, incendio y partículas finas recibidos del Proteus mostrados en el Node Dashboard.

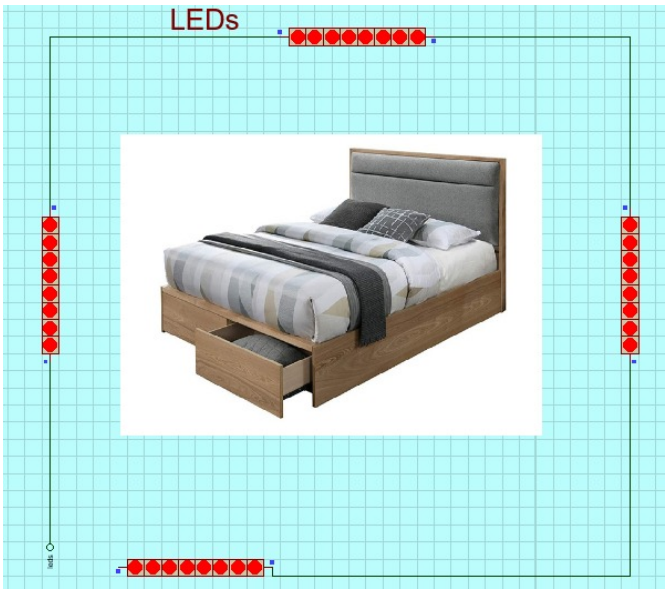
En base al color del foco que esté siendo prendido, el color de la luz en el cuarto cambiará.


LEDs

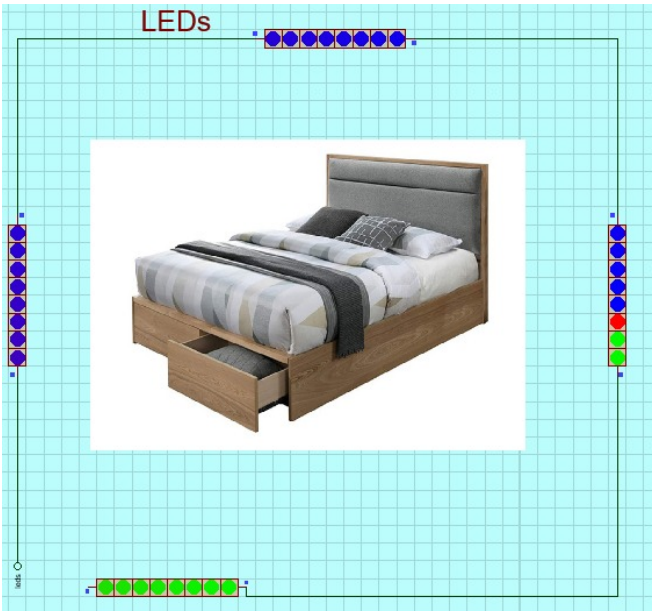

Verde


LEDs


Red


LEDs


Rainbow


LEDs

5. Resultados

La simulación del sistema se llevó a cabo en múltiples computadoras con el fin de comprobar su correcto funcionamiento. Asimismo, se repitió múltiples veces para cerciorarnos de que los dispositivos estén funcionando correctamente y mostrando las lecturas correspondientes. En esa misma línea, las luces de los LEDS fueron capaces de responder a las lecturas atípicas que son capaces de poner en riesgo la seguridad de los integrantes del hogar "virtual". En ese sentido, con ello se completó uno de los objetivos del proyecto.

Por otro lado, tambien se ha corroborado la capacidad de Node-Red para ser capaz de mostrar las lecturas correctas a través de un Dashboard, el cuál es mas amigable para el usuario que la lectura en una consola. Asimismo, se ha demostrado que con la creación de un cliente, nuestro sistema es capaz de enviar mensajes a nuestro usuario a través de la Web y mantenerlo constantemente informado acerca de la integridad de su hogar.

También es importante destacar que se encontraron dificultades varias a lo largo de la implementación de este proyecto. En primer lugar, en muchas oportunidades, en la lectura de estímulos, el microcontrolador mostraba lecturas érroneas que se fueron subsanando con los días. De igual forma, también se encontraron dificultades al momento de realizar la conexión de Node-Red debido a un tema de librería y del servidor. Concluyendo, el presente trabajo ha demostrado que puede ser correctamente establecido para mantener la seguridad del hogar, y para ser implementado de forma física a un bajo costo.

6. Conclusiones

En conclusión, nuestro proyecto de dispositivo para monitorear la calidad del aire, la temperatura y la humedad, y la cantidad de partículas en el aire en ambientes cerrados ha

demostrado ser efectivo en detectar la presencia de gases tóxicos en casos de incendio y de mantener un ambiente cómodo y saludable para las personas. Asimismo, también se hace un enfoque incisivo en el tema de la seguridad, dado que también se incluyó un sensor ultrasónico. En cuanto a los resultados obtenidos en Arduino IDE y, posteriormente, en Node-Red muestran que el dispositivo es capaz de medir y registrar de manera precisa los niveles de contaminación del aire y la temperatura y humedad del ambiente, lo que puede ayudar a fortalecer la seguridad de nuestra hogar. Además, el uso de sensores de bajo costo y la implementación en un microcontrolador Arduino hacen que este dispositivo sea accesible y fácil de construir para la mayoría de las personas. En resumen, creemos que este proyecto tiene el potencial de hacer una contribución importante en el campo de la salud y el bienestar en ambientes cerrados.

7. Bibliografía

BBC News. (2021, November 2). Delhi 'most polluted city' in the world, says WHO. Retrieved from https://www.bbc.com/news/world-asia-india-59138781

Chen, Y. R., Ko, T. H., Lee, Y. Y., & Lin, J. N. (2019). A low-cost indoor air quality monitoring system for ambient PM2.5 and PM10 detection. Sensors, 19(21), 4649. https://doi.org/10.3390/s19214649

NASA. (2019, November 19). Amazon Fires Consumed Nearly 40 Million Acres in 2019. Retrieved from https://www.nasa.gov/feature/goddard/2019/amazon-fires-consumed-nearly-40-million-acres-in-2019

Rajashekara, K., Hegde, M., & Ramanathan, G. (2021). Smart Indoor Air Quality Monitoring System for Early Warning of Sick Building Syndrome. Journal of Green Building, 16(1), 1-16. https://doi.org/10.3992/jgb.16.1.1

Yang, W., Ren, Y., Liu, Y., Shi, J., & Zhou, W. (2021). Sistemas de monitoreo de calidad del aire en interiores: tecnologías, desafíos y perspectivas. Building and Environment. https://doi.org/10.1016/j.buildenv.2021.107865

8. Anexos

8.1. Código en Arduino

```
//Agregar librerias necesarias
#include <Adafruit_NeoPixel.h>
#include <DHT.h>                //Para el sensor DHT
#include <DHT_U.h>              //Para el sensor DHT

//Declaracion de variables globales
int Gas = 9;                    //Pin del sensor de gas
int redLed = 7;                 //Pin del led asociado al sensor de gas
int partisensor = 2;            //Pin del sensor de particulas finas
int partiLed = 4;               //Pin del led asociado a particulas finas
int TRIG = 10;                  //Pin del trigger en el puerto 10
int ECO = 11;                   //Pin del eco en el puerto 11
const int analogInluz = A5;     //Pin analogico de entrada para el sensor de luz
const int analogOutluz = 8;     //Pin analogico de salida para el sensor de luz

int sensorValue = 0;        //Valor obtenido del sensor de luz
int outputValue = 0;        //Valor que se muestra del sensor de luz
String luz;                 //Variable para el uso del foco
int tiempo;                 //Variable para calcular el tiempo en el sensor de ultrasonido
int dis;                    //Variable para calcular la distancia en el sensor de ultrasonido
String alarmaSeg;           //Variable para la alarma del sensor de ultrasonido
String alarmaIn;            //Variable para la alarma del sensor de gas
String alarmaPart;          //Variable para la alarma del sensor de particulas finas
String temperatura;         //Variable para calcular la temperatura en el dht
String humedad;             //Variable para calcular el porcentaje de humedad en el dht
char input;                 //Variable para la entrada de datos por el monitor serial

//Definiciones de variables
#define DHTPIN  12
#define DHTTYPE DHT11
#define PIN     3           //Pin
#define N_LEDS  32          //Numero de leds

DHT_Unified dht(DHTPIN, DHTTYPE);
Adafruit_NeoPixel strip = Adafruit_NeoPixel(N_LEDS, PIN, NEO_GRB +
NEO_KHZ800);


//SET UP    --- Codigo a ejecutarse una vez
void setup() {
  Serial.begin(9600);         //Puerto serial con velocidad de transmision de 9600 baudios
```

```
  dht.begin();                    //Inicia el sensor dht
  strip.begin();                  //Inicia el strip
  pinMode(Gas, INPUT);            //Establece como entrada el pin del sensor de gas
  pinMode(TRIG, OUTPUT);          //Establece como salida el pin del trigger
  pinMode(ECO, INPUT);            //Establece como entrada el pin del eco
  pinMode(partiLed, OUTPUT);    //Establece como salida el pin del led asociado al sensor de
particulas finas
  pinMode(partisensor, INPUT);  //Establece como entrada el pin del sensor de particulas finas

  sensor_t sensor;                        //Crea una variable sensor para el sensor dht
  dht.temperature().getSensor(&sensor);  //Para obtener la temperatura del sensor dht
  dht.humidity().getSensor(&sensor);      //Para obtener la humedad del sensor dht
 }



//LOOP    --- Codigo a ejecutarse continuamente
void loop() {
  //Sensor de luz ----------------------------------------------------------------------------
  sensorValue = analogRead(analogInluz);          //Lectura del sensor de luz
  outputValue = map(sensorValue, 0, 1023, 0, 255);  //Mapea el valor de salida
  analogWrite(analogOutluz, outputValue);          //Asignarlo al rango de la salida analógica

  if (outputValue <= 200){          //Si el valor de salida (en esta caso luz) es menor a 200
    luz = "1";                              //Se activa el foco
  }
  else {                                  //Caso contrario
    luz = "0";                              //Se mantiene apagado
  }


  //Ultrasonido ------------------------------------------------------------------------------
  digitalWrite(TRIG, HIGH);       //Emisión del pulso
  delay(1);                       //Retraso de 1 ms
  digitalWrite(TRIG, LOW);        //Se deja de emitir el pulso
  tiempo = pulseIn(ECO, HIGH);  //Con función pulseIn se espera un pulso
  dis = 1 + tiempo / 58.2;          //Ecuación para convertir el valor de duracion en una
distancia dada en cm

  if(dis <= 50){              //Si la distancia es menor a 50
    alarmaSeg = "1";                //La alarma se activara
  }
  else {                      //Caso contrario
    alarmaSeg = "0";                //Se mantendra apagada
  }


  //Gas --------------------------------------------------------------------------------------
```

```
        if(digitalRead(Gas) == HIGH){//Si la lectura del gas es ALTA (haciendo referencia a fuga)
          alarmaIn = "1";                   //Se activara la alarma
          digitalWrite(6, LOW);             //Se apagara el led del pin 6
          digitalWrite(7, HIGH);            //Se prendera el led del pin 7
        }
        else {                              //Caso contrario
          alarmaIn = "0";                 //La alarma permanecera apagada
          digitalWrite(6, HIGH);            //Se prendera el led del pin 6
          digitalWrite(7, LOW);             //Se apagara el led del pin 7
        }


        //Temperatura humedad ------------------------------------------------------------------------
        //Humedad
        sensors_event_t event;              //Crea una variable sensor para el sensor dht
        dht.humidity().getEvent(&event);    //Para obtener la humedad del sensor dht
        humedad = event.relative_humidity;  //Se asigna ese valor a otra variable

        //Temperatura
        dht.temperature().getEvent(&event); //Para obtener la temperatura del sensor dht
        temperatura = event.temperature;    //Se asigna ese valor a otra variable
        delay(10);                          //Retraso de 10 ms


        //Partículas finas ---------------------------------------------------------------------------
        int particulas = pulseIn(partisensor,LOW);      //Obtiene la cantidad de particulas
        float ratio = particulas / 1000.0;              //Calcula el ratio (en milisegundos)
        //Se convierte (en este caso se aproximo lo mejor posible)
        int concentration =
((1.1*pow(ratio,3)-3.8*pow(ratio,2))/5)+((1.1*pow(ratio,3)-3.8*pow(ratio,2))/500)-((1.1*pow(ratio,3)
-3.8*pow(ratio,2))/35); // calcular la concentración de partículas en el aire (en pcs/0.01cf)

        if (concentration < 1000){      //Si la concentracion es menor a 1000
          alarmaPart = "1";                 //Se activara la alarma
          digitalWrite(partiLed, HIGH);     //Se prendera el led del pin partiLed definido
previamente
        }
        else {                          //Caso contrario
          alarmaPart = "0";                 //Se apagara la alarma
          digitalWrite(partiLed, LOW);      //Se apagara el led del pin partiLed definido
previamente
        }
        delay(10);                      //Retraso de 10 ms


        //Impresion en el monitor serial
        Serial.println(alarmaIn + ", " + alarmaSeg + ", " + luz + ", " + temperatura + ", " + humedad
+ ", " + alarmaPart);
```

//Impresion que se realizaba previamente, pero tuvo que cambiarse para hacer mas sencilla la parte de Node-Red

```
/*
Serial.println("--------------------------------------------------------------------------------");
Serial.println("                          Estado de los sensores");
Serial.println();

Serial.print("   Alarma de fuga de gas: ");
if(alarmaIn == "0"){
  Serial.println("Apagada");
}
else {
  Serial.println("Encendida");
}

Serial.print("   Puerta: ");
if(alarmaSeg == "0"){
  Serial.println("Cerrada");
}
else {
  Serial.println("Abierta");
}

Serial.print("   Foco: ");
if(luz == "0"){
  Serial.println("Apagado");
}
else {
  Serial.println("Encendido");
}

Serial.println("   Temperatura: " + temperatura);
Serial.println("   Humedad: " + humedad);

// Esto estaba comentado
// Serial.print("Ratio : ");
// Serial.println(ratio);
// Serial.print("Duracion : ");
// Serial.println(particulas);

Serial.print("   Alarma de Particulas finas por litro (pcs/L): ");
if(alarmaPart == "0"){
  Serial.println("Apagada");
}
```

```
  else {
    Serial.println("Encendida");
  }
  Serial.println("--------------------------------------------------------------------------------");
  Serial.println();
  Serial.println();
  Serial.println();
  */

  //Encender leds ---------------------------------------------------------------------------
  while (Serial.available() > 0){  //Mientras la entrada del monitor serial sea mayor a 0
    input = Serial.read();                  //Se leera el valor ingresado
    Serial.println(input);                  //Se imprimira dicho valor
    if(input == '0'){                       //Si el valor es 0
      chase(strip.Color(0, 0, 0));                  //Estaran apagados
    }
    if(input == '1'){                       //Si el valor es 1
      chase(strip.Color(0, 0, 255));                //Seran de color azul
    }
    if(input == '2'){                       //Si el valor es 2
      chase(strip.Color(0, 255, 0));                //Seran de color verde
    }
    if(input == '3'){                       //Si el valor es 3
      chase(strip.Color(255, 0, 0));                //Seran de color rojo
    }
    if(input == '4'){                       //Si el valor es 4
      rainbow(30);                                  //Seran de los colores del arcoiris
    }
  }
  delay(1000);                              //Retraso de 1 s
}



//Funciones para leds
static void chase(uint32_t c){
  for(uint16_t i = 33; i > 0; i--){
    strip.setPixelColor(i - 1, c);  //Dibuja un nuevo pixel
    strip.show();
    delay(30);
  }
}

void rainbow(uint8_t wait){
  uint16_t i, j;

  for(j = 0; j < 256; j++){
    for (i = 0; i < strip.numPixels(); i++){
```

```
      strip.setPixelColor(i, Wheel((i*1+j) & 255));
    }
    strip.show();
    delay(wait);
  }
}

//Ingresa un valor de 0 a 255 para obtener un valor de color
//Los colores son una transición r-g-b de vuelta a r (rojo)
uint32_t Wheel(byte WheelPos){
  if (WheelPos > 85){
    return strip.Color(WheelPos * 3, 255 - WheelPos * 3, 0);
  }
  else {
    if (WheelPos <170){
      WheelPos -= 85;
      return strip.Color(255 - WheelPos * 3, 0, WheelPos * 3);
    }
    else {
      WheelPos -= 170;
      return strip.Color(0, WheelPos * 3, 255 - WheelPos * 3);
    }
  }
}
```

8.2. Base de datos

```
//Crea la base de datos
CREATE DATABASE iot_project;


//Uso de esta base de datos para no tener inconvenientes con las ya creadas
USE iot_project;


//Crea la tabla para el sensor de particulas
CREATE TABLE particulate_sensor (
   id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
   value FLOAT(10, 2),
   timestamp TIMESTAMP
```

```
);

//Crea la tabla para el sensor de humo
CREATE TABLE smoke_sensor (
    id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    value FLOAT(10, 2),
    timestamp TIMESTAMP
);

//Crea la tabla para el sensor de temperatura y humedad
CREATE TABLE temp_humidity_sensor (
    id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    temperature FLOAT(10, 2),
    humidity FLOAT(10, 2),
    timestamp TIMESTAMP
);

//Crea la tabla para el sensor ultrasonidp
CREATE TABLE ultrasonic_sensor (
    id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    value FLOAT(10, 2),
    timestamp TIMESTAMP
);
```

```
//Crea la tabla para el led
CREATE TABLE led (
    id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    color VARCHAR(20),
    state TINYINT(1),
    timestamp TIMESTAMP
);
```