ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

IMPLEMENTACIÓN DE UN SISTEMA DE MONITOREO DE AMENAZAS FÍSICAS EN EL CUARTO DE TELECOMUNICACIONES DEL NODO GOSSEAL DE TELCONET

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE TECNÓLOGO EN ELECTRÓNICA Y TELECOMUNICACIONES

DARIO FRANCISCO IBAÑEZ VELASTEGUI

dario.ibanez@epn.edu.ec

PAUL ADRIAN SILVA SOLIS

paul.silva@epn.edu.ec

DIRECTOR: Ing. FANNY PAULINA FLORES ESTÉVEZ MSc.

fanny.flores@epn.edu.ec

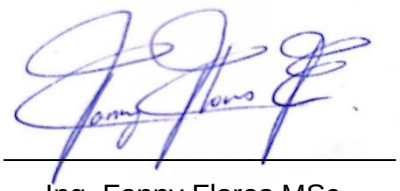
DIRECTOR: ING. MÓNICA DE LOURDES VINUEZA RHOR MSc.

monica.vinueza@epn.edu.ec

Quito, agosto 2020

CERTIFICACIÓN

Certificamos que el presente trabajo fue desarrollado por Dario Francisco Ibañez Velastegui y Paul Adrian Silva Solis, bajo nuestra supervisión.



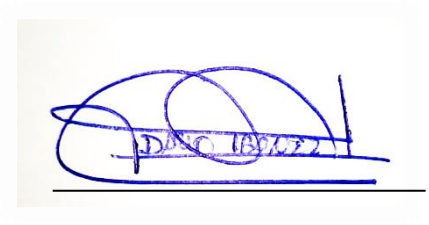
Ing. Fanny Flores MSc.
Director del Proyecto

Ing. Mónica Vinueza Rhor MSc.
Codirector del Proyecto

DECLARACIÓN

“Nosotros, Dario Francisco Ibañez Velastegui y Paul Adrian Silva Solis declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría, que no ha sido previamente presentado para ningún grado o calificación profesional; y, que hemos consultado las referencias bibliográficas que se incluyen en este documento.

Sin perjuicio de los derechos reconocidos en el primer párrafo del artículo 114 del Código Orgánico de la Economía Social de los Conocimientos, Creatividad e Innovación -COESC-, somos titulares de la obra en mención y otorgamos una licencia gratuita, intransferible y no exclusiva de uso con fines académicos a la Escuela Politécnica Nacional. Entregaremos toda la información técnica pertinente. En el caso de que hubiese una explotación comercial de la obra por parte de la EPN, se negociará los porcentajes de los beneficios conforme lo establece la normativa nacional vigente".



Dario Francisco Ibañez Velastegui

Paul Adrian Silva Solis

DEDICATORIA

Dedico este proyecto de titulación a Dios por permitirme haber llegado a este momento importante de mi carrera universitaria. A mi madre por apoyarme y demostrarme su cariño en toda mi vida. A mi esposa por ser uno de los pilares importantes en mi vida y por demostrarme siempre su amor. A mis hermanos por las palabras de aliento y apoyo incondicional. A mi tío, a quien quiero como un padre, por compartir momentos significativos conmigo y por guiarme cuando mi padre falleció. A mi padre y abuelita, que ya no están a mi lado, pero su amor prevalece en mi corazón, y a pesar de la distancia que nos separa, siento que están conmigo siempre y aunque nos faltaron muchas metas por alcanzar juntos, tengo la certeza que estarían muy orgullosos de mí en este momento.

Dario

Dedico este proyecto de titulación a todas aquellas personas que me apoyaron moral y económicamente.

Paul

AGRADECIMIENTO

Agradezco a Dios por las bondades dadas, por permitirme tener una familia grandiosa y una esposa maravillosa, que supieron apoyarme en toda mi vida, brindándome su amor y cariño durante mi carrera universitaria. Gracias por creer en mí. De igual manera a la Escuela De Formación De Tecnólogos de la Escuela Politécnica Nacional, por haberme permitido formarme en ella, a los ingenieros que fueron partícipes de este proceso, especialmente a la tutora académica Ingeniera Fanny Flores, quien con su paciencia, confianza y rectitud nos ha guiado durante el proyecto. Finalmente, a Telconet S.A. empresa líder de telecomunicaciones en Ecuador, en especial a los Ingenieros Santiago Jayo y Jorge Zambrano, por habernos permitido realizar el proyecto de titulación dentro de una de las instalaciones de la empresa. Gracias por la confianza y apoyo durante este tiempo.

Dario

Quiero expresar mi más sincera gratitud a cada una las personas que creyeron en mí, especialmente a mi familia por el apoyo brindado en este trayecto, a mis amigos, a la Escuela de Formación de Tecnólogos de la Escuela Politécnica Nacional por haberme permitido formarme en sus aulas y a aquellos buenos docentes que me brindaron lo mejor de sí mismos. A Telconet S.A por su apoyo al permitirnos sus instalaciones para desarrollar este proyecto de titulación.

Paul

ÍNDICE DE CONTENIDO

CERTIFICACIÓN I
DECLARACIÓN II
DEDICATORIA III
AGRADECIMIENTO IV
ÍNDICE DE CONTENIDO V
ÍNDICE DE FIGURAS VII
ÍNDICE DE TABLAS IX
RESUMEN X
ABSTRACT XI
1. INTRODUCCIÓN 1
1.1. Marco teórico 2
Elementos del Sistema 2
Hardware 2
Plataforma de *middleware* 14
Herramientas de visualización e interpretación 18
2. METODOLOGÍA 21
3. RESULTADOS Y DISCUSIÓN 22
3.1. Inspección en el nodo Gosseal 22
3.2. Diseño del proyecto 23
Hardware 23
Software 32
3.3. Implementación del sistema 34
Instalación de canalización 34
Instalación de cableado de datos y eléctrico 36
Elaboración de *cases* 36
Elaboración de PCB 37
Instalación de dispositivos 37
3.4. Desarrollo del *software* 39
Configuración de Arduino 39
Desarrollo de la API REST 43
Base de Datos 48
Desarrollo de la página *web* 50
3.5. Pruebas de funcionamiento 59
Pruebas de encendido del sistema 59
Pruebas de sensores 60

Prueba de funcionamiento de la API 64
Prueba de comunicación entre Arduino y la base de datos. 67
Prueba de funcionamiento de la página *web* 71
Prueba de funcionamiento del sistema integrado 73
4. CONCLUSIONES Y RECOMENDACIONES 75
4.1. Conclusiones 75
4.2. Recomendaciones 77
5. REFERENCIAS BIBLIOGRAFÍCAS 79
ANEXOS ..... **¡Error! Marcador no definido.**

ÍNDICE DE FIGURAS

Figura 1.1: Elementos del sistema 2
Figura 1.2: *Raspberry PI* 3B+ 3
Figura 1.3: Dispositivos de almacenamiento MicroSD 4
Figura 1.4: Arduino UNO 5
Figura 1.5: *Ethernet Shield* 6
Figura 1.6: Sensor DHT22 7
Figura 1.7: Estructura interna del sensor de humedad 8
Figura 1.8: Curva característica del termistor NTC 9
Figura 1.9: Sensor flama 5 canales 11
Figura 1.10: Sensor magnético 12
Figura 1.11: Ejemplo de funcionamiento REST 15
Figura 1.12: Estructura Básica *FRONT- END* 19
Figura 3.1: Esquema general del sistema 23
Figura 3.2: Simbología planos 23
Figura 3.3: Diagrama nodo Gosseal 24
Figura 3.4: Diagrama cuarto de UPS 25
Figura 3.5: Diagrama área del generador 25
Figura 3.6: Diagrama de red nodo *Gosseal* 26
Figura 3.7: Conexión entre Arduino del nodo y sensores 27
Figura 3.8: Conexión entre Arduino del cuarto de UPS y sensores 27
Figura 3.9: Conexión entre Arduino del generador y sensores 28
Figura 3.10: Circuito de acondicionamiento para sensores SCT 29
Figura 3.11: Entrada y salida de la señal de voltaje del sensor de corriente 29
Figura 3.12: Circuito de acondicionamiento para 6 sensores 30
Figura 3.13: PCB cara inferior y superior 30
Figura 3.14: *Case* Arduino nodo 31
Figura 3.15: *Case* Arduino cuarto de UPS 31
Figura 3.16: *Case* Arduino generador 32
Figura 3.17: Diagrama de flujo del sistema 33
Figura 3.18: Canalización 34
Figura 3.19: Tramo final 34
Figura 3.20: Caja de derivación tipo FS redonda 35
Figura 3.21: Cajetín de datos 35
Figura 3.22: Punto eléctrico 36
Figura 3.23: *Case* del Arduino del generador 36

Figura 3.24: PCBs terminadas 37
Figura 3.25: *Case* del Arduino de nodo terminado 37
Figura 3.26: Instalación de sensores SCT 38
Figura 3.27: Arduino área del generador 38
Figura 3.28: Estrcutura de la base de datos 49
Figura 3.29: Tabla de valores de sensores de Temperatura 49
Figura 3.30: Vistas del modelo MVC del proyecto 52
Figura 3.31: Plantilla *blank page Admin* LTE 57
Figura 3.32: Toma de valores DHT 60
Figura 3.33: Toma de valores MQ2 61
Figura 3.34: Mensaje de alerta ante la presencia de humo 61
Figura 3.35: Toma de valores SCT-013 62
Figura 3.36: Toma de valores sensor de flama 63
Figura 3.37: Toma de valores sensor magnético 64
Figura 3.38: Inicio del servidor 65
Figura 3.39: Valores de temperatura registrados en la base de datos 65
Figura 3.40: Mensajes de alerta registrados en la base de datos 66
Figura 3.41: Valores de humedad desde la base de datos 66
Figura 3.42: Valores de voltaje desde la base de datos 67
Figura 3.43: Envío de información a la API 67
Figura 3.44: Peticiones GET de todas las URLs 68
Figura 3.45: URL: http://192.168.100.48:5000/SensorData/temperatura 68
Figura 3.46: Tabla de corriente 69
Figura 3.47: Tabla de gas 69
Figura 3.48: Tabla de humedad 70
Figura 3.49: Tabla de mensajes 70
Figura 3.50: Tabla de temperaturas 71
Figura 3.51: Tabla de voltajes 71
Figura 3.53: Página de bienvenida 72
Figura 3.54: Página de corrientes 73
Figura 3.55: Página de lectura de temperatura en tiempo real 74
Figura 3.56: Página de lectura de gas en tiempo real 74

ÍNDICE DE TABLAS

Tabla 1.1: Características de la *Raspberry PI* 3B+ 3
Tabla 1.2: Sensores y tipo de Aplicación 7
Tabla 1.3: Especificaciones técnicas del sensor 8
Tabla 1.4: Gases que puede detectar el sensor MQ2 9
Tabla 1.5: Especificaciones técnicas del sensor MQ2 10
Tabla 1.6: Características técnicas de los sensores SCT 11
Tabla 1. 7: Características técnicas del sensor de flama 5 canales 11
Tabla 1.8: Tipos de datos predefinidos que utiliza SQL 16
Tabla 3.1: Tabla temperatura de la base de datos 50
Tabla 3.2: Valores de temperatura y humedad medidos por el DHT 60
Tabla 3.3: Valores de concentración de gas en el ambiente 62
Tabla 3.4: Valores de corriente medidos por el SCT-013 62
Tabla 3.5: Valores medidos por el sensor de flama 63
Tabla 3.6: Valores medidos por el sensor magnético 64

RESUMEN

El presente proyecto describe el proceso de implementación de un sistema de monitoreo de amenazas físicas para la empresa Telconet en una de sus minicentrales de telecomunicaciones denominada “Nodo Gosseal”. El sistema de monitoreo es una red LAN dentro de la intranet del nodo, conformada por varios Arduinos y una Raspberry Pi. Al tener varios Arduinos dentro de un mismo sistema, se puede obtener información de varios sensores de distinto tipo ubicados en varias áreas dentro de la central de telecomunicaciones.

La sección uno contiene una introducción, así como la justificación en donde se detalla la finalidad del proyecto. Además, se revisa la teoría concerniente a los diferentes componentes del sistema en la parte de *hardware* y de *software*.

La segunda sección describe la metodología utilizada para la ejecución del proyecto mediante el estudio de amenazas físicas distribuidas existentes en el Nodo Gosseal.

En la tercera sección se describe los resultados y discusiones obtenidos al implementar un sistema de monitoreo basado en sensores, dispositivos de control (Arduino) y dispositivos de almacenamiento (*Raspberry* Pi). Luego de cada implementación, se realiza pruebas de funcionamiento y de comunicación.

La cuarta sección muestra las conclusiones y recomendaciones obtenidas a partir de la implementación y el funcionamiento del sistema de monitoreo de amenazas físicas en la minicentral de telecomunicaciones “nodo *Gosseal*”.

Finalmente, se presenta las referencias bibliográficas que soportan el desarrollo del proyecto; así como los anexos que incluyen el certificado de funcionamiento del proyecto, plano del sistema de monitoreo de amenazas, manual de usuario y los códigos implementados.

Palabras Clave: Centro de datos, sistema de monitoreo, sensores, aplicación *web, apirest.*

ABSTRACT

This project describes the process of implementing a system for monitoring physical threats for the company Telconet in one of its mini telecommunication centers denominated "Gosseal Node". The monitoring system is a LAN within the node's intranet, made up of several Arduinos and a Raspberry Pi. Having assorted Arduinos within the same system, it is possible to obtain information from several sensors of different types located in various areas inside the telecommunication center.

The first section contains an introduction, as well as the justification where the purpose of the project is detailed. In addition, the theory concerning the different components of the system is reviewed in the hardware and software parts.

The second section describes the methodology used for the execution of the project through the study of existing distributed physical hazards in the Gosseal Node.

The third section describes the results and discussions obtained by implementing a monitoring system based on sensors, control devices (Arduino) and storage devices (Raspberry Pi). After each implementation, functional and communication tests are performed.

The fourth section shows the conclusions and recommendations obtained from the implementation and operation of the physical threat monitoring system in the mini telecommunication center "Gosseal node".

Finally, the bibliographical references that support the development of the project are presented, as well as the annexes that include the certificate of operation of the project, map of hazard monitoring system, Manual of User and the implemented codes.

Keywords: *Data center, monitoring system, sensors, web application, restapi.*

1. INTRODUCCIÓN

En la actualidad, los diseños de centros de datos, grandes salas de cómputo y nodos alojan equipos de telecomunicaciones como *switches, routers*, sistemas de almacenamiento, sistemas de respaldo, servidores, entre otros. En consecuencia, dichos lugares deben contar con las mejores condiciones físicas y ambientales para preservar su entorno, mismas que deben estar definidas bajo estándares técnicos. Además, se debe considerar la importancia de que, tanto la temperatura como la humedad sean las adecuadas, incluir dispositivos para detección de contaminantes peligrosos y para detección de incendios [1].

Los equipos físicos actuales como Sistema de Alimentación Ininterrumpida (SAI) o en inglés (UPS), sistemas de aire acondicionado o equipos modernos de telecomunicaciones, cuentan con funciones de monitoreo y alerta sofisticadas que, en muchos casos, no son suficientes para detectar amenazas en forma global. Entre las amenazas físicas más frecuentes se puede mencionar: problemas en la alimentación de los equipos, temperatura, incendios y calidad del aire [2].

La instrumentación industrial es el conjunto de herramientas (equipamientos y dispositivos) que utilizan los ingenieros o técnicos para medir, convertir, registrar y transmitir variables de un proceso. Estas variables pueden ser físicas o químicas [3] [4].

Telconet cuenta con equipos de telecomunicaciones y sistemas de climatización. Sin embargo, el aumento de suscriptores al servicio de Internet, ha ocasionado que los equipos de telecomunicaciones aumenten en número y de igual manera en productividad. Esto ha ocasionado que con el pasar del tiempo, se detecte puntos críticos.

El sistema de monitoreo de amenazas físicas en el nodo *Gosseal* de Telconet, constituirá una herramienta de gran utilidad, ya que permitirá visualizar alertas a los operarios del nodo ante cualquier amenaza, informando en forma rápida y oportuna de eventos que se estén suscitando. Esto beneficiará a la empresa, preservando la vida útil de los equipos al evitar temperaturas elevadas, casos de incendios o humo, como también acumulación de electricidad estática en los puntos de baja humedad.

1.1. Marco teórico

Elementos del Sistema

Hay tres componentes o elementos básicos que interactúan entre sí; en la figura 1.1 se observa un diagrama esquemático de estos componentes.

1. *Hardware*: son objetos inteligentes como sensores, actuadores y otros dispositivos de comunicación.
2. Plataforma de *middleware*: es el *software* que permite el intercambio de información entre las aplicaciones, así como las herramientas computacionales que permiten el análisis de datos.
3. Herramientas de visualización e interpretación: es el *software* que permite el monitoreo y reporte de la información y que deben ser diseñadas para ser accesadas por diferentes aplicaciones y dispositivos [5].

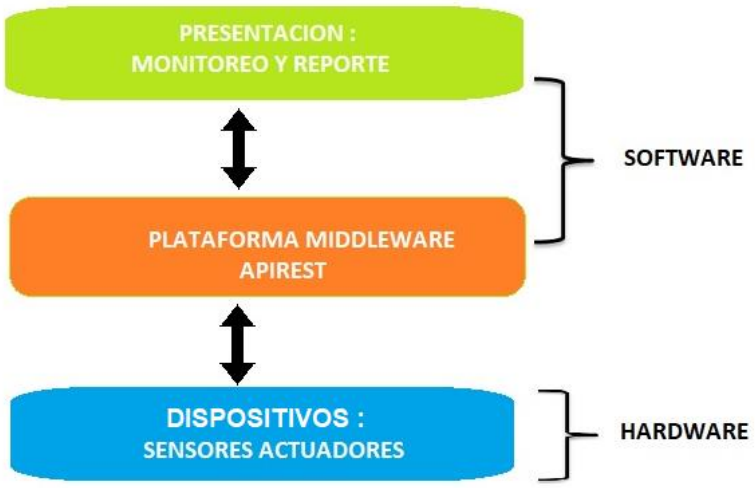


Figura 1.1: Elementos del sistema [5]

Hardware

- ***Raspberry PI* 3 B+**

La *Raspberry Pi* 3B+ es una computadora de tamaño reducido mucho más rápida, multiplataforma, que permite la instalación de una gran variedad de sistemas operativos tales como, *Noobs*, *Ubuntu y Windows*, aunque el más utilizado suele ser *Raspbian* basado en *Debian*. En la Tabla 1.1 se presenta las características más importantes de *Raspberry* Pi 3B+ [6] [7].

Tabla 1.1: Características de la *Raspberry PI* 3B+ [8]

CARACTERÍSTICAS	RASPBERRY PI 3B+
Procesador	Broadcom BCM2837B0, Cortex-A53 64-bit SoC @ 1.4GHz
Memoria RAM	1GB LPDDR2 SDRAM
USB	4 puertos USB 2.0
Conexión a Internet	Gigabit *Ethernet* sobre USB 2.0 (300 Mbps)
Wifi	2.4GHz y 5GHz, estándar IEEE 802.11.b/g/n/ac *Wireless* LAN
Bluetooth	4.2
Pines GPIO	40
Video y Sonido	1 puerto HDMI conector CSI camera conector DSI camera
Tarjeta SD	Formato Micro SD para cargar el sistema operativo y almacenamiento de datos
Temperatura de funcionamiento	0 – 50ºC
Alimentación	5v/2.5 A vía conector micro USB 5V DC vía pines GPIO *Power over Ethernet* (PoE)

El GPIO (*General Purpose Input/Output*), es un sistema de 40 pines de entrada / salida de propósito o uso general, estos pines son interfaces físicas entre la *Raspberry Pi* y el mundo exterior. Los pines pueden activarse o desactivarse mediante interrupciones. Los 40 pines están divididos en 2 filas de 20 pines cada una, 28 de los 40 pines son GPIO; estos a su vez están compuestos por 17 pines GPIO normales para uso en proyectos y 11 pines GPIO especiales para uso como puerto UART, I2C o SPI. Los 12 restantes son de alimentación (5V o 3.3V) o tierra. En la figura 1.2 se puede apreciar la Rasberry PI 3 modelo B+.

Figura 1.2: *Raspberry PI* 3B+ [7]

Interfaz de red *Ethernet:* La *Raspberry Pi* 3B+ cuenta con una interfaz de red 1000 Base T. Usa un concentrador *Gigabit Ethernet*, el *microchip* LAN7515 que puede alcanzar velocidad real de 300 Mbps, debido a la interfaz USB2.0 entre el concentrador (LAN7515) y el procesador principal. Con la velocidad de *Ethernet* mejorada, se ha implementado *PoE* en la placa. Con el módulo *PoE HAT* es posible alimentar la Raspberry Pi 3B+ a través de *Ethernet* [9].

Almacenamiento: La *Raspberry Pi* 3 B+ no dispone de un dispositivo de almacenamiento propio como un disco duro o un dispositivo de almacenamiento masivo USB, cuenta con un lector o ranura para memorias microSD y 4 puertos USB 2.0. Al lector de tarjetas se puede conectar una tarjeta micro SD de al menos 2 GB de capacidad para que albergue todo el sistema operativo. Otra opción es usar dispositivos USB de Almacenamiento Masivo (UMS – USB *Mass Storage*), conectado mediante un cable USB a cualquiera de los 4 puertos USB 2.0 disponibles. Entre ellos se puede utilizar:

- Discos duros físicos.
- Unidades de estado sólido (SSD)
- Memorias USB de bolsillo

La microSD también denominada dispositivo de arranque, es usada como un dispositivo de almacenamiento principal. La ventaja de la placa consiste en que casi todos los dispositivos USB de almacenamiento masivo pueden ser leídos. En la figura 1.3 se presenta una microSD de almacenamiento masivo [10] [11].

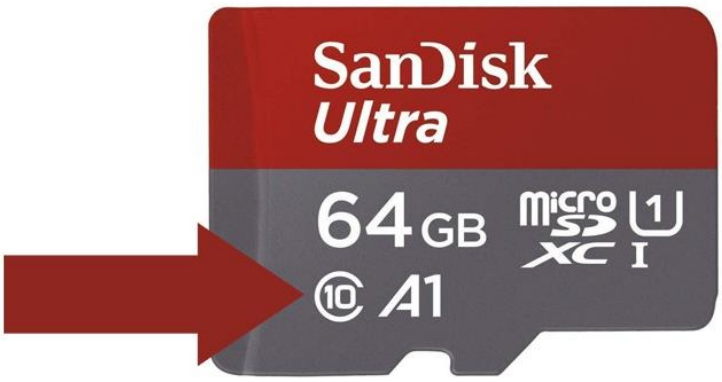


Figura 1.3: Dispositivos de almacenamiento MicroSD [12]

- **Arduino UNO**

Arduino UNO R3 es la placa más utilizada entre los usuarios, debido a su diseño, versatilidad y especialmente bajo costo. El Arduino UNO es un sistema construido con el microcontrolador de 8 bits ATmega328 de Atmel, sencillo y de bajo costo. Físicamente tiene 14 pines enumerados del 0 a 13 que pueden ser configurados como entradas o salidas digitales. Los pines 2 al 7 pueden ser configurados como salidas PWM (*Pulse Width Modulation*), lo que permite variar el ciclo de trabajo de la señal cuadrada. En el pin 13 está conectado un LED que se utiliza como dispositivo de salida en la verificación y depuración de programas. Los pines 0 y 1 se pueden configurar para utilizar como un puerto serie.

El microcontrolador Atmega328 posee un conversor analógico-digital (A/D) de 6 canales enumerados del A0 al A5, con una resolución de 10 bits, retornando un valor entero entre 0 y 1023. Estos pines también pueden ser usados como pines digitales, aunque el principal uso es para la lectura de sensores analógicos. Cuenta con una memoria *flash* de 32 Kb, memoria SRAM de 2 Kb y memoria EEPROM de 1Kb. La placa Arduino puede ser alimentada a través de un cable USB conectado a un computador, fuente de alimentación externa o baterías. El voltaje recomendado de alimentación es de 7 a 9 Vcd. La ubicación física de los pines en la placa Arduino se puede apreciar en la figura 1.4 [13] [14].

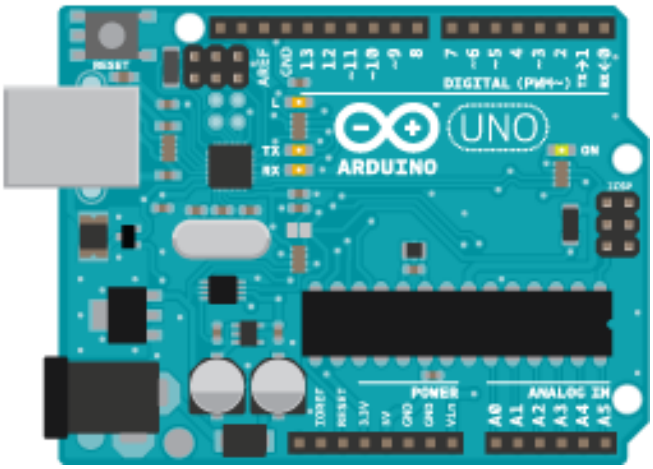


Figura 1.4: Arduino UNO [13]

- ***Ethernet Shield***

Las *shields* son placas de circuito impreso que se pueden conectar en la parte superior de la placa Arduino mediante acoplamiento de los pines, extendiendo sus características sin perder el número de pines. *Ethernet Shield* permite que la placa Arduino UNO se conecte a una red

cableada utilizando el protocolo TCP/IP, este *shield* cuenta con el *chip* controlador W5100 y se configura usando la librería por defecto del lenguaje de Arduino "*Ethernet*". Este *shield* se conecta con la placa Arduino Uno, por la parte de arriba mediante los pines. La *shield* contiene un *jack* estándar RJ45 que puede ser conectado a un switch o enrutador, utilizando un cable de red estándar CAT5e o CAT6, o utilizar un cable de red cruzado para conectarlo directamente al computador.

El voltaje de alimentación es de 5VCD que, se los toma de la placa Arduino. Los pines 10, 11, 12 y 13 sirven para comunicar el *microchip* W5100 y la placa Arduino UNO (vía SPI). También cuenta con un zócalo para colocar una tarjeta microSD, la cual se podrá utilizar usando la librería por defecto del lenguaje de Arduino "SD"; los pines 4, 11, 12 y 13 sirven para que la tarjeta microSD se pueda comunicar con la placa Arduino UNO. En la figura 1.5 se puede observar la *shield Ethernet*.

No se pueden utilizar los dos dispositivos a la vez debido a que comparten el canal SPI. Además, tiene la posibilidad de acoplamiento de un módulo PoE. Físicamente, dispone de un botón de "*reset*", el cual reinicia tanto el *chip* W5100 y la placa Arduino; y LEDs informativos que indican cierta función:

- PWR: indica que la placa y el *shield* están encendidos.
- LINK: indica la presencia de una conexión de red, y parpadea cuando el *shield* recibe o transmite datos.
- FULLD: indica que la conexión de red es "*full duplex*".
- "100M": indica la presencia de una conexión de red de 100 Mbps en lugar de una de 10 Mbps.
- RX: parpadea cuando el *shield* recibe datos.
- TX: parpadea cuando el *shield* envía datos.
- COLL: parpadea cuando se detectan colisiones de paquetes en la red [15] [16].

Figura 1.5: *Ethernet Shield* [13]

- **Sensores**

Un sensor es un dispositivo que puede captar una determinada acción externa. En otras palabras, los sensores imitan la capacidad de percepción de los seres humanos. Se puede encontrar sensores relacionados con los sentidos de los seres humanos:

- Vista: sensor reacciona a la luz.
- Oído: sensor reacciona al sonido.
- Tacto: sensor reacciona al contacto.
- Olfato: sensor reacciona a los olores naturales (mezclas de especies químicas que tienen miles de constituyentes) [17].

La tabla 1.2 muestra el tipo de sensor más utilizado en función de su aplicación.

Tabla 1.2: Sensores y tipo de Aplicación [18]

SENSORES	APLICACIÓN
Termistores (PTC, NTC, Semiconductores)	Temperatura
Resistivos, Capacitivos	Humedad
Magnéticos, Infrarrojos	Presencia
Infrarrojos	Distancia / Flama
Detectores de gases y Humos	Químicos

Sensor de temperatura y humedad relativa DHT22: El sensor DHT22 que se puede apreciar en la figura 1.6, es un sensor digital para obtener mediciones de temperatura y humedad relativa (RH). Este tipo de sensor viene precalibrado de fábrica. Posee un microcontrolador que recibe, procesa y convierte la señal analógica en señal digital calibrada dando así, calidad y fiabilidad en las lecturas. El sensor posee una interfaz serie propietario, que solo requiere de un pin para comunicarse con un microcontrolador [19].

Figura 1.6: Sensor DHT22 [19]

La tabla 1.3 muestra las características técnicas de los sensores DHT22.

Tabla 1.3: Especificaciones técnicas del sensor DHT22 [20]

CARACTERÍSTICA	VALORES
Tensión de funcionamiento	3 a 5V
Corriente de funcionamiento máxima	2.5mA
Rango de humedad	0-100% / 2-5%
Rango de temperatura	-40 a 80 ° C / ± 0.5 ° C
Tasa de muestreo	0.5 Hz (lectura cada 2 segundos)
Ventaja	preciso

El DHT22 físicamente está compuesto por dos sensores resistivos, un sensor de humedad junto con un sensor de temperatura NTC (o termistor). El sensor resistivo de humedad, está compuesto por dos electrodos con sustrato (sal o un polímero plástico conductor) intercalado entre ellos que retiene la humedad, como se puede apreciar en la figura 1.7. El sustrato a medida que absorbe el vapor de agua libera iones, lo que a su vez aumenta la conductividad entre los electrodos. Las variaciones de resistencia entre los dos electrodos son proporcionales a la humedad relativa. La resistencia entre los electrodos disminuye cuando se detecta una humedad relativa alta y aumenta cuando se detecta una humedad relativa baja.

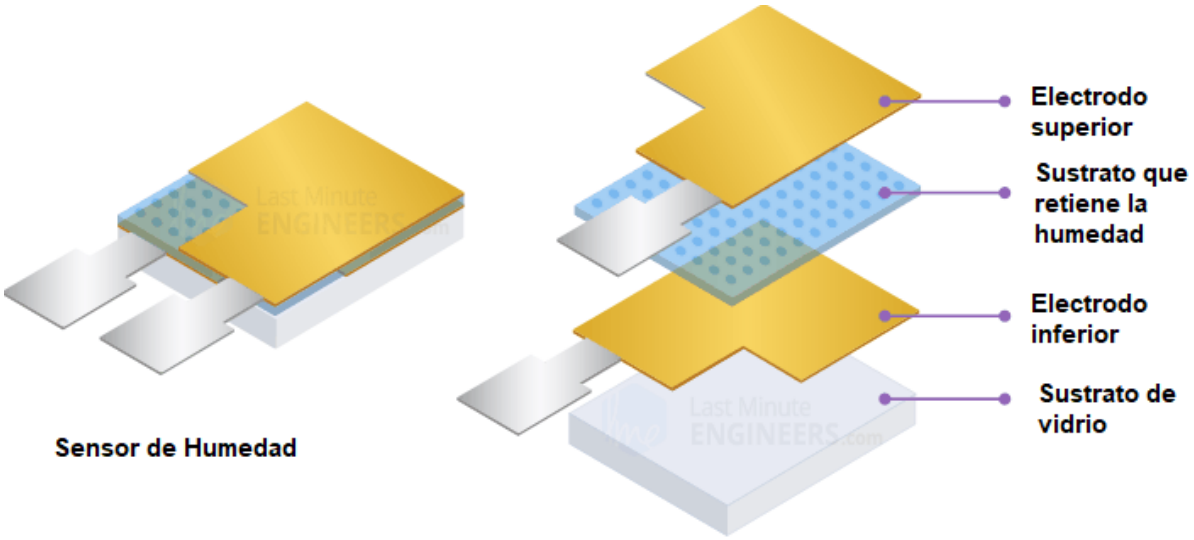


Figura 1.7: Estructura interna del sensor de humedad [19]

Los sensores de temperatura NTC (o termistores), son resistencias construidas de la mezcla y sinterización de óxidos dopados de manganeso (Mn), níquel (Ni), cobalto (Co), cobre (Cu) y Hierro (Fe); fabricados en varios tamaños y tipos de encapsulados en cerámica. Los

termistores se pueden conectar directamente a la entrada de un convertidor A/D de un microcontrolador cualquiera. La resistencia de un NTC a 25º C va desde los 50Ω hasta varios MΩ. Los sensores de temperatura funcionan como un termómetro, captan la temperatura del entorno y los cambios de temperatura son mostrados como variaciones de voltaje. En la figura 1.8, se puede observar que, a medida que la temperatura aumenta la resistencia del termistor NTC disminuye y viceversa [19].

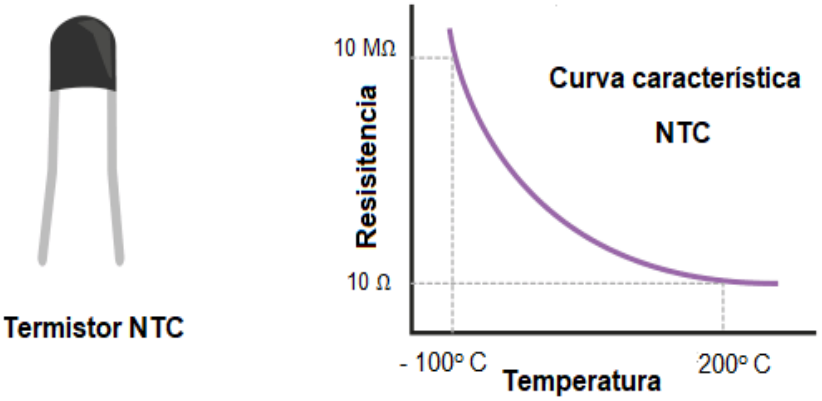


Figura 1.8: Curva característica del termistor NTC [19]

Sensor de gas MQ2: Los sensores de gas de la serie MQ son los más utilizados para proyectos con Arduino; esto se debe a su bajo costo, alta sensibilidad, respuesta rápida, capacidad de medir una gran cantidad de sustancias, fácil conexión y uso. En la tabla 1.4 se muestra los gases que puede detectar este sensor [21].

Tabla 1.4: Gases que puede detectar el sensor MQ2 [21]

GASES QUE PUEDE DETECTAR	RANGO DE DETENCIÓN (PPM)
Hidrógeno	300 – 500
LGP y Propano	200 - 500
Metano	5000 - 20000
Monóxido de carbono	No especificado
Alcohol	100 - 2000

Los sensores de gas son transductores entre la reacción química de un gas y una resistencia de óxido de semiconductor, el cambio de resistividad en el material hace que ocurra la reacción. Los sensores de gas necesitan 3 elementos principales:

- Película sensora (película de óxido semiconductor).
- Microcalefactor.
- Circuito de acondicionamiento.

Cuando un gas es detectado, se produce un cambio en la resistividad del óxido semiconductor, que ocasiona que el circuito de lectura registre un cambio en la resistencia equivalente. Dicho cambio está relacionado con la concentración de un determinado gas en el ambiente [22]. La tabla 1.5 muestra las principales características técnicas de los sensores de gas MQ2.

Tabla 1.5: Especificaciones técnicas del sensor MQ2 [23]

CARACTERÍSTICA	VALORES
Tensión de funcionamiento	4.5V a 5V DC
Temperatura de trabajo	-20 ℃ ~ +55 ℃
Humedad: ≤ 95% RH	≤ 95% RH
Rango de detección	300 a 10000 ppm
Tiempo de precalentamiento	24 horas

Sensor de corriente alterna no invasivo (SCT-013 y SCT-019): Los sensores SCT permiten medir la corriente que circula por un conductor, sin la necesidad de intervenir (abrir, cortar) el circuito para la instalación. Estos sensores son trasformadores de corriente que utilizan la inducción electromagnética para su funcionamiento. Por sus características, se lo usa para monitorear consumos de energía y equipos de medición de CA. Los sensores SCT se componen de tres partes:

- Devanado primario (1 espira): Conductor (cable) en el cual se desea saber la corriente que circula por él.
- Núcleo: Por lo general elaborado de ferrita y por el cual debe atravesar el conductor.
- Devanado secundario: internamente dentro del sensor cuyas espiras pueden alcanzar un máximo de 200, dependiendo del modelo [24].

Los sensores SCT vienen fabricados con 2 tipos de salida (salida de corriente y salida de voltaje), y en diferentes capacidades. La tabla 1.6 muestra las características técnicas de los sensores SCT utilizados en el proyecto [25].

Tabla 1.6: Características técnicas de los sensores SCT [25] [26]

CARACTERÍSTICA	SCT -013	SCT-019
Corriente de entrada	0 -100 A	0 -200 A
Corriente de salida	0 -50 mA	0 -33 mA
Temperatura de trabajo	-25ºC a 70ºC	-25ºC a 70ºC
Material del núcleo	Ferrita	*Permalloy*
Propiedad resistencia al fuego	de acuerdo con UL94-V0	de acuerdo con UL94-V0

Módulo Sensor de flama IR 5 canales: Este sensor puede detectar llamas debido a su estructura LED detectora de fuego. Trabaja dentro de la banda del infrarrojo del espectro electromagnético con un rango de longitudes de onda de 760 nm a 1110 nm, valores correspondientes a la mayoría de las llamas, incendios e incluso puede detectar algún tipo de luz demasiado potente, como una lámpara incandescente o el mismo sol. Este módulo sensor está formado de 5 sensores YG1006 y el comparador LM393, como se puede apreciar en la figura 1.9. Adicional, cuenta con un potenciómetro que permite calibrar la sensibilidad del módulo YG1006. Al estar compuesto por 5 sensores individuales, el rango de detección de la flama aumenta [27] [28]. La tabla 1.7 muestra las características técnicas de los sensores de flama 5 canales.

Figura 1.9: Sensor flama 5 canales [28]

Tabla 1. 7: Características técnicas del sensor de flama 5 canales [28]

CARACTERÍSTICA	SENSOR 5 CANALES
Longitud de onda	700nm ~ 1100nm
Tipo de salida	Análoga y digital
Voltaje de alimentación	3.3V – 9V
Rango de detección	120º
Sensor	Diodo IR de 5 mm - YG1006

Sensor magnético*:* Es un interruptor eléctrico que para ser activado necesita de un campo magnético, que puede ser generado por un imán permanente o por una bobina. Físicamente, está compuesto de un par de láminas ferromagnéticas, herméticamente selladas en una cápsula de vidrio, como se puede apreciar en la figura 110 [29]. Los elementos de un interruptor magnético son:

- Interruptor
- Imán de accionamiento [30]

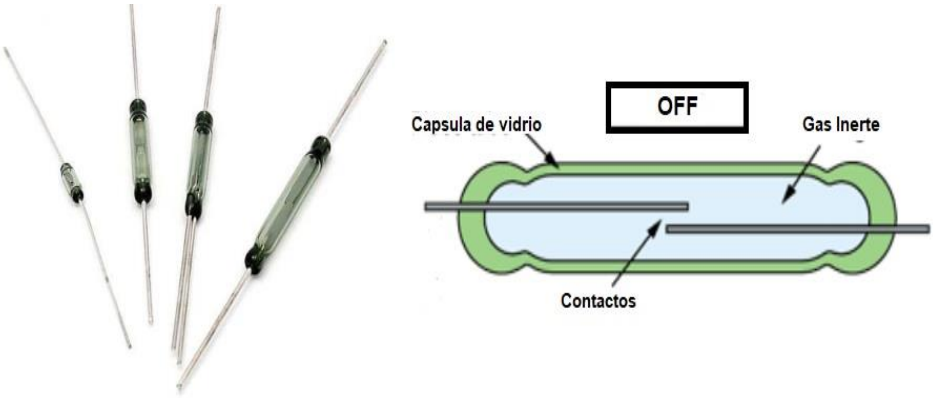


Figura 1.10: Sensor magnético [29]

- **Cableado estructurado**

ANSI/TIA 568-D

Esta Norma regula todo lo concerniente a sistemas de cableado estructurado para edificios comerciales. La serie de normas ANSI/TIA 568-D incorpora lo siguiente:

- Cables horizontales reconocidos.
- 4 pares de 100Ω no blindado o blindado de par trenzado balanceado, Categoría 3, 5e, 6 y 6A.
- Radio de curvatura del par trenzado balanceado es 4 veces el diámetro del cable.
- Fuerza de tracción máxima: 25lb.
- Radio de curvatura del cable de fibra: 10 veces el diámetro del cable.

Conectores Reconocidos:

- 8 posiciones RJ45 conector modular y terminación T568A o T568B

Cableado horizontal topología:

- Cada conexión de salida en el área de trabajo tiene un tramo de cable individual.

- Topología en estrella.
- Todo enlace de cable horizontal se limita a 90 metros de longitud.
- Usar el 40 % de llenado máximo para vías de cable [31].

ANSI/TIA 569

Los Enrutamientos horizontales incluyen:

- Ducto bajo el piso
- Piso Falso
- Tubo *conduit*
- Bandeja de cables
- Enrutamientos de techo falso
- Canaletas (rutas perimetrales)

Tubo *Conduit:* Es un ducto cerrado que proporciona espacios y trayectorias para la instalación de los cables de telecomunicaciones. También se toma en cuenta lo siguiente:

- Un solo tubo dará servicio a 3 salidas como máximo y deberá aumentar de tamaño desde la salida más alejada hacia el TC.
- Ninguna sección será más larga de 30 metros o contendrá más de dos curvaturas de 90 grados entre los cajetines de paso.
- El radio de curvatura interno en un conducto igual o menor de 2'' será por lo menos 6 veces el diámetro interno del tubo.
- Si es mayor a 2'' deben tener un radio mínimo de curvatura de 10 veces el diámetro.
- El grado de relleno máximo para conductos será del 40% [31].

Cajetines de Paso: Cumplen la función de facilitar la instalación y el halado de cables o conductores.

- Ubicado en una recta y accesible en una sección de la canaleta.
- No debe ser usado para empalmar cables, o en lugar de una curvatura [31].

ANSI/TIA 606

Provee un esquema de administración uniforme. Las áreas a ser administradas son:

- Terminaciones
- Medio de transmisión
- Enrutamiento

La información debe ser presentada mediante:

- Etiquetas
- Registros
- Reportes
- Planos
- Órdenes de trabajo
- Actas de entrega

Plataforma de *middleware*

Se puede definir como un *software* de conexión que se encuentra entre el sistema operativo y la aplicación. Es utilizado para la interacción y ejecución de múltiples procesos en distintas máquinas a través de la red, en base a un conjunto de servicios que pueden ser internos o externos que resuelven problemas de heterogeneidad entre sistemas operativos. A nivel de aplicación, permite la interacción entre programas en un ambiente distribuido [32].

- ***Apirest***

Es una API que utiliza la arquitectura REST para comunicarse con las demás aplicaciones utilizando verbos http [33].

Application Programming Interface **(API):** Interfaz de programación de aplicaciones; se define como un conjunto de subrutinas, funciones y procedimientos para que un programa (aplicación) pueda comunicarse con otro. Cuando dos aplicaciones requieren comunicarse entre sí, se utiliza una API que abstrae la lógica de los sistemas; es decir, para la comunicación los programas exponen interfaces. El programa A envía los parámetros necesarios para que la aplicación B realice los procesos obligatorios para que registre, actualice o borre alguna información sin manipular la base de datos; tanto el programa A y B siguen siendo independientes [34].

Representational State Transfer **(REST):** El cliente y servidor tendrán una comunicación constate para lo cual, se deberá establecer reglas de comunicación entre los dos. *Rest* es una arquitectura que instaura un conjunto de limitaciones arquitectónicas y que establece cómo se realiza la comunicación entre dos equipos, que tratan de minimizar la latencia y las comunicaciones de la red y; al mismo tiempo, maximizar la independencia y escalabilidad de las implementaciones de los componentes.
Las seis restricciones de REST son:

- Cliente-servidor
- Sin estado
- Caché
- Interfaz uniforme
- Sistema en capas
- Código según la demanda (opcional) [35]

REST funciona de la siguiente manera; un servidor tiene un recurso y un cliente puede solicitar una "representación" (descripción) de los recursos (datos). Por ejemplo, si el recurso es la información almacenada en una base de datos, entonces su "estado de representación" podría ser una simple lista de los valores en el registro de la base de datos, como se puede apreciar en la figura 1.11.

Figura 1.11: Ejemplo de funcionamiento REST [36]

- ***Raspbian***

Raspbian es un sistema operativo gratuito desarrollado en Debian, optimizado para el *hardware* Raspberry Pi. *Raspbian* contiene más de 35,000 paquetes, *software* precompilado incluido en un formato agradable para una fácil instalación en la *Raspberry Pi* [37].

- **Base de Datos**

Una base de datos es un conjunto estructurado de datos conocida como metadatos, que representa entidades y sus interrelaciones. La representación será única e integral; a pesar de que debe permitir, utilizaciones varias y simultáneas. Las bases de datos relacionales se

basan en la organización de la información en trozos pequeños. Son independientes de la aplicación; al realizar modificaciones no destructivas en la estructura, no se afecta la aplicación. La estructura se basa en la relación o tabla, junto con la habilidad de definir relaciones complejas entre ellas; a las cuales, se puede acceder directamente sin la lentitud de las limitaciones de los modelos jerárquicos o propietario / miembro, que requieren de una navegación a través de una estructura compleja de datos [38].

- **SQL (*Structured Query Language*)**

SQL es el lenguaje estándar ANSI/ISO de definición, manipulación y control de bases de datos relacionales. Es un lenguaje declarativo; solo hay que indicar qué se quiere hacer, a diferencia de los lenguajes procedimentales, en los que es necesario especificar la manera de realizar cualquier acción sobre la base de datos. SQL es considerado un lenguaje muy expresivo y parecido al lenguaje natural (inglés), Por estas razones, SQL se ha convertido en el lenguaje estándar para manejo de base de datos relacionales, con el que se puede acceder a todos los sistemas relacionales comerciales. Las bases de datos son un conjunto de tablas; dichas tablas están compuestas por columnas a las que se asigna un tipo de dato predefinido [39]. La tabla 1.8 muestra el tipo de datos predefinidos que utiliza SQL.

Tabla 1.8: Tipos de datos predefinidos que utiliza SQL [39]

TIPO DE DATO	DESCRIPCIÓN
BIT	Cadenas de bits de longitud fija.
INTEGER	Números enteros.
FLOAT	Números con coma flotante con la precisión especificada.
DOUBLE	Números con coma flotante con más precisión predefinida que la del tipo REAL.
DATE	Fechas, compuestas por: año, mes, día.
TIME	Tiempo, compuesto por: hora, minutos, segundos.

- **Entorno virtual**

Un entorno virtual es un ambiente apartado que es creado con el objetivo de aislar recursos como librerías y entornos de ejecución del sistema principal o de otros entornos virtuales; es decir, que en un mismo sistema es posible tener instaladas múltiples versiones de una misma librería sin crear ningún tipo de conflicto. Los entornos virtuales fueron desarrollados con el fin de poder utilizar diferentes versiones de un mismo paquete en una sola máquina al estar desarrollando diferentes programas sin crear ningún tipo de conflicto [40].

- ***Python***

Python es *open source*, por lo que está en un proceso de continuo desarrollo por una gran comunidad de desarrolladores de la organización *Phyton Software Foundation*. Python es un lenguaje de alto nivel interpretado y de multipropósito. Es multiplataforma, por lo que puede ser utilizado en cualquier sistema operativo. Tiene licencia *Python Software Foundation License.*

Las características de *Python* son:

- Lenguaje interpretado: a diferencia de otros lenguajes, *Python* no necesita compilarse, *Python* utiliza un intérprete que se encarga de leer el fichero fuente y ejecutarlo.
- Multiplataforma: el intérprete puede ser utilizado en diferentes plataformas y sistemas operativos.
- Tipado Dinámico: no es necesario declarar el tipo de variable.
- Código legible: facilita la lectura y comprensión de código de terceros.
- Multiparadigma: permite trabajar con programación estructurada, funcional, orientada a objetos [41].

- **Paquetes de Instalación**

Una dependencia es un paquete requerido por otro programa para que funcione correctamente.

Flask*:* Es un *framework* ligero escrito en *Python* que permite crear aplicaciones *web* de forma rápida y escribiendo un número reducido de líneas de código. Utiliza el motor de plantillas Jinja y del *kit* de herramientas Werkzeug WSGI y tiene una licencia BSD [42].

Flask-HTTPAuth*:* es una extensión para *Flask* que simplifica el uso de autenticación HTTP con rutas de *Flask* [43].

***Flask-RESTful*:** es una extensión para *Flask* que agrega soporte para el desarrollo de una API REST en forma rápida y sencilla, funciona con bibliotecas ORM existentes. *Flask-RESTful* impulsa las mejores prácticas con una configuración mínima [44].

Flask-SQLAlchemy: es una extensión para *Flask* que agrega soporte para *SQLAlchemy* a la API. El objetivo de la extensión es simplificar el uso de *SQLAlchemy* con *Flask,*

proporcionando valores predeterminados útiles que facilitan la realización de tareas comunes [45].

Flask-Marshmallow: es una delgada extensión que permite la integración de *Flask* y *marshmallow,* agregando características adicionales a *marshmallow,* que incluyen los campos de URL e hipervínculos para las API. Opcionalmente, se puede integrar *Flask-SQLAlchemy* [46].

Marshmallow: es una biblioteca ORM / ODM para convertir datos complejos como objetos, hacia y desde tipos de datos nativos de Python [47].

SQLAlchemy: es un conjunto de herramientas Python SQL y *Object Relational Mapper* (ORM), que proporciona a los desarrolladores de APIs potencia y flexibilidad de SQL [48].

Marshmallow-Sqlalchemy: integración de SQLAlchemy con la biblioteca de serialización de *marshmallow* [49].

- **Librerías Arduino**

Las librerías son fragmentos de códigos hechas por otras personas para aumentar la funcionalidad de un programa. Existen dos maneras de instalar una librería en Arduino; ya sea, directamente desde la aplicación o a través de un archivo ZIP. También se puede usar las bibliotecas que vienen por defecto en el IDE o crear librerías propias [13].

Herramientas de visualización e interpretación

- ***Front-end y Back-end***

El *front-end* proporciona la interfaz con el usuario final. Constituye la parte del *software* que interactúa con el o los usuarios. El *back-end* es la parte que procesa las solicitudes realizadas por los usuarios desde el *front-end*. De esta manera, *front-end* y b*ack-end* interactúan en un sistema *web* o *software* para resolver las necesidades de los usuarios**,** como se puede apreciar en la figura 1.12. La finalidad de esta abstracción es mantener separadas las diferentes partes de un sistema *web* o *software* (*front-end* recolecta los datos y el *back-end* los procesa) con el fin de tener un mejor control [50].

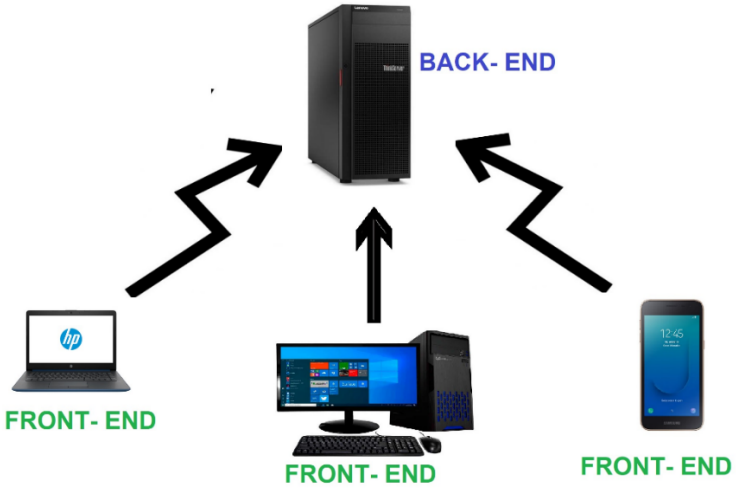


Figura 1.12: Estructura Básica *FRONT- END* [50]

- **Modelo-vista-controlador**

MVC es una arquitectura de *software*, que se presenta frecuentemente en aplicaciones *web*. MVC separa los datos de una aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos:

- Modelo
- Vista
- Controlador

Modelo: Representa la parte fuerte de la aplicación, toda la información con lo que el sistema opera. Se encarga de ejecutar o de llevar a cabo los servicios o eventos dictados por el controlador y gestionar la base de datos (consultar, actualizar, borrar, buscar). Representa los datos del programa. El sistema es el encargado de mantener las relaciones pertinentes entre el modelo y la vista, también se encarga de avisar a la vista cuando cambie el modelo [51].

Vista: Es la encargada de generar la representación visual del modelo en el cual se mostrarán los datos; es la interfaz que se genera y que llega al usuario, a través de la cual interactúa y envía peticiones [51].

Controlador: Es el encargado de recibir las diferentes peticiones del usuario, inicializando/invocando los servicios o eventos correspondientes a dicha petición. Con los

datos obtenidos, devuelve un modelo a la capa de vista que posteriormente se encargará de hacer llegar al usuario [51].

- ***Bootstrap***

Es un *kit* de herramientas multiplataforma *open source* para desarrollar sitios y aplicaciones *web* utilizando HTML, CSS y JS [52].

- **HTML**

Hyper Text Markup Language o HTML, es un lenguaje de marcado, utilizado por los programadores con el objetivo de redactar instrucciones que los ordenadores interpretan y ejecutan para generar una página *web*. HTML usa etiquetas que contienen las instrucciones necesarias para que el navegador pueda interpretar cómo mostrar los elementos de la página. HTML es un lenguaje básico para construir páginas *web,* pero necesario para los programadores [53].

- ***Admin* LTE**

Es una de las plantillas más usadas para administración y desarrollo de sitios *web.* Constituye una plantilla HTML que usa *Bootstrap* y *jQuery*. *Admin* LTE es un proyecto *open source* con licencia MIT. Se puede descargar en 2 versiones; la versión "*Ready*" que descarga el archivo compilado y listo para usar y la versión "*Source code*" que descarga el código fuente para personalizar de acuerdo con las necesidades del usuario [54].

- ***Postman***

Es un *software* que ayuda al desarrollo de APIs. El uso más común es para el testeo de APIs. También ayuda a documentar, monitorear, simular las API. Es de fácil instalación [55].

- ***Sqlite Browser***

Es un *software open source* que funciona como herramienta visual de alta calidad para visualizar, crear, diseñar y editar archivos de bases de datos compatibles con SQLite [56].

2. METODOLOGÍA

Este proyecto fue realizado en base en a la investigación aplicada. Tal investigación se centra en cómo se pueden llevar los conocimientos adquiridos de forma teórica hacia la práctica. La motivación de esta investigación aplicada es la resolución de los problemas que se plantean en un momento dado. El desarrollo del monitoreo de amenazas ayuda con el control de diferentes dispositivos del cuarto de telecomunicaciones en el nodo *Gosseal* de Telconet, permitiendo que el personal autorizado mantenga una lectura instantánea como: voltaje, corriente, temperatura y llevar un historial de control de acceso en el cuarto de telecomunicaciones.

Para la implementación del Control de Monitoreo de amenazas físicas se procedió de la siguiente manera: se efectúo un levantamiento de información sobre los equipos de alta prioridad en el cuarto de telecomunicaciones para la administración y realización de un diseño de la ubicación de cada sensor con respecto a su equipo. Seguidamente, se estructuró un diseño del nodo con sus medidas acercándose a lo real; los sensores, el cableado de datos y el cableado eléctrico fueron considerados en el diseño del proyecto.

Posteriormente, se procedió a ejecutar la implementación de la tubería, tanto para el cableado eléctrico como el cableado de datos. Para esto se tomó en cuenta como referencia: las normas y estándares para centros de datos y cuartos de telecomunicaciones.

Finalmente, se evaluó el rendimiento de cada uno de los sensores, tomando lecturas y rangos para la correcta recolección de datos. Mediante un modelo Cliente-Servidor, se creó un servidor *web* con lenguaje de programación *Python* donde está alojada la base de datos. Este servidor *web* está alojado en un *host* (*Raspberry*). En el servidor *web* está desarrollada una *API REST* que permitió la comunicación entre el Cliente y el Servidor, siendo el cliente cada Arduino. Los resultados que se encuentren en la base de datos se visualizan en una aplicación *web*.

3. RESULTADOS Y DISCUSIÓN

3.1. Inspección en el nodo *Gosseal*

Telconet ha creado varias mini centrales de telecomunicaciones denominadas nodos, uno de estos nodos está ubicado en la ciudad de Quito, sector Quito Tenis, calle Pedro Gosseal #148. Siendo esta una de las primeras mini centrales, no cuenta con un sistema de monitoreo eficaz o control automático de sus equipos. Este proyecto surgió desde la perspectiva de mejorar aquellas falencias.

Se realizó una visita técnica en conjunto con el personal encargado del funcionamiento de la mini central de telecomunicaciones *Gosseal.* La visita se realizó con el propósito de constatar las amenazas físicas distribuidas y escuchar los requerimientos de la administración de Telconet para la implementación del monitoreo de amenazas en el nodo. Se pudo constatar las siguientes novedades:

- No existía un dispositivo ni sistema de monitoreo de amenazas físicas en el cual se refleje los valores en tiempo real.
- No existía un sistema de monitoreo de corriente en el nodo donde se refleje los valores de las líneas provenientes de EEQ, como en el generador.
- No se contaba con detectores de flama apuntados a equipos que trabajan con alta corriente en el cuarto de UPS y generador.
- En equipos prioritarios no existía ningún control de temperatura y humedad externo que indicara si el aire a su alrededor estaba en un nivel elevado o normal.
- Existen 3 áreas dentro de la infraestructura del nodo *Gosseal* en donde se debe instalar el sistema de monitoreo, estas son: el nodo, el cuarto de UPS y área del generador.

Tomando estos factores, se planteó junto a la administración encargada del nodo en Telconet, un sistema de monitoreo de amenazas físicas como: temperatura, humedad, detección de humo, detección de incendios, monitoreo de corriente y voltaje y detección de apertura de puerta. Esto permitirá a la empresa seguir con su plan de mejoras, haciendo posible que se pueda tomar las medidas necesarias a tiempo, para solventar problemas rápidamente y no afecten la operatividad de la red.
En la figura 3.1 se muestra un esquema general del sistema.

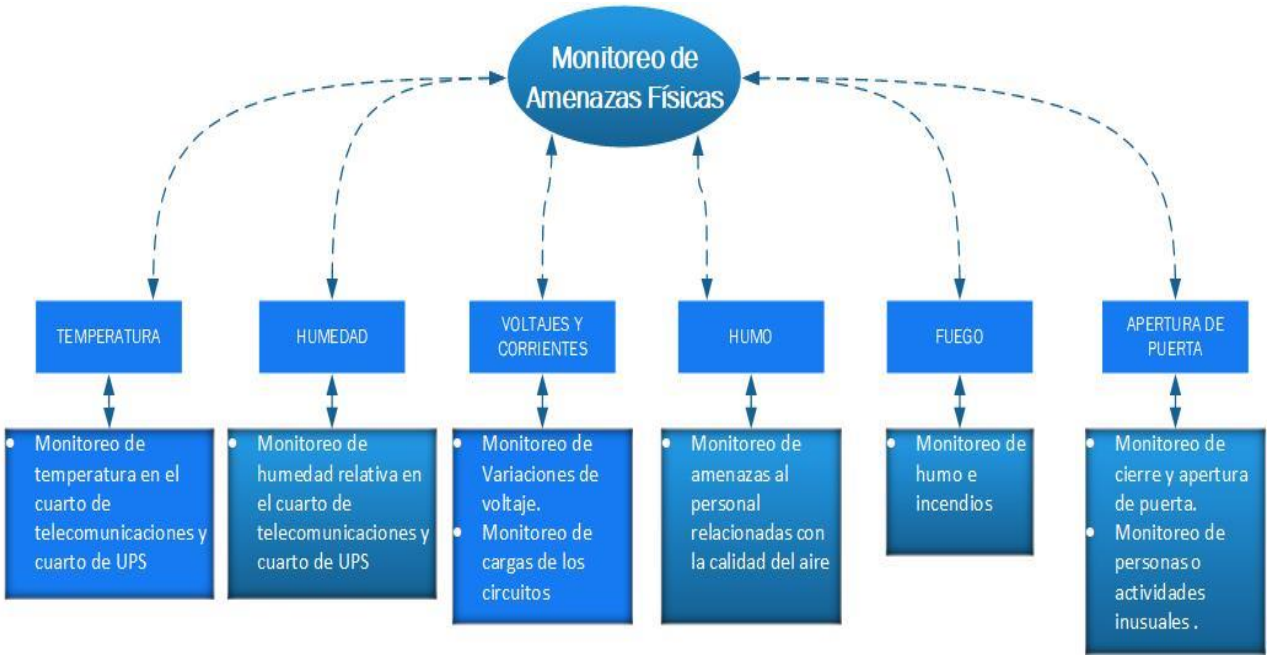


Figura 3.1: Esquema general del sistema

3.2. Diseño del proyecto

Hardware

- **Cableado estructurado y conexión de sensores**

Se procedió a tomar las medidas físicas del nodo, estas medidas ayudaron a elaborar el diagrama. Se utilizó la herramienta AutoCAD, para posicionar los sensores en cada una de las áreas del nodo, conjuntamente con el cableado de datos y el cableado eléctrico. Este diseño fue aprobado por la administración de Telconet y facilitó la autorización del pedido de materiales. En la figura 3.2 se presenta la simbología utilizada en los planos.

Elementos	Descripcion	Elementos	Descripcion
I	SENSOR DE CORRIENTE		TOMA CORRIENTES
F	SENSOR DE FLAMA		CAJETÍN RJ45
G	SENSOR DE GAS		TUBERÍA 3/4 PARA CABLE DE DATOS
	CASE DE ARDUINO		TUBERÍA 1/2 CABLE ELÉCTRICO
	SENSOR DHT22		SENSOR MAGNETICO

Figura 3.2: Simbología planos

En las figuras 3.3, 3.4 y 3.5 se presenta los planos de las áreas del nodo *Gosseal* donde fue instalado el sistema de monitoreo. En los planos se puede visualizar tanto la distribución de los sensores como de cableado.

Como se muestra en la Figura 3.3, en el nodo *Gosseal*, un Arduino UNO está ubicado de manera central con respecto a los sensores. Los sensores de temperatura y humedad (DHT22) están ubicados en las áreas de equipos considerados de alta prioridad, el sensor de gas MQ2 está ubicado junto al Arduino para que funcione como detector complementario al que Telconet tiene instalado y el sensor magnético está ubicado en la en la puerta de acceso al nodo principal.

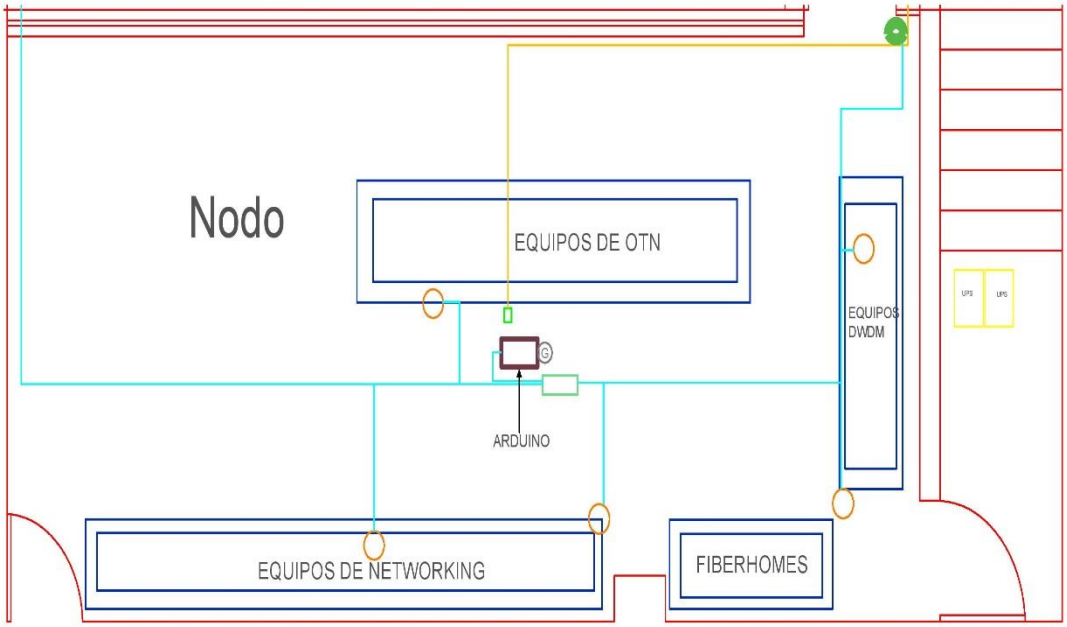


Figura 3.3: Diagrama nodo Gosseal

Como se muestra en la Figura 3.4, en el cuarto de UPS, un Arduino está ubicado de manera central con respecto a los dos sensores de temperatura y humedad; sin embargo, está a poca distancia de los UPS, esto con el propósito de monitorear voltaje y corriente de los mismos.

Como se muestra en la Figura 3.5, en el área del generador, un Arduino está ubicado sobre el tablero de trasferencia automática (T.T.A); esto con el propósito de monitorear voltajes y

corrientes, tanto de las líneas de la Empresa Eléctrica, como voltajes y corrientes de las líneas del generador cuando este encienda.

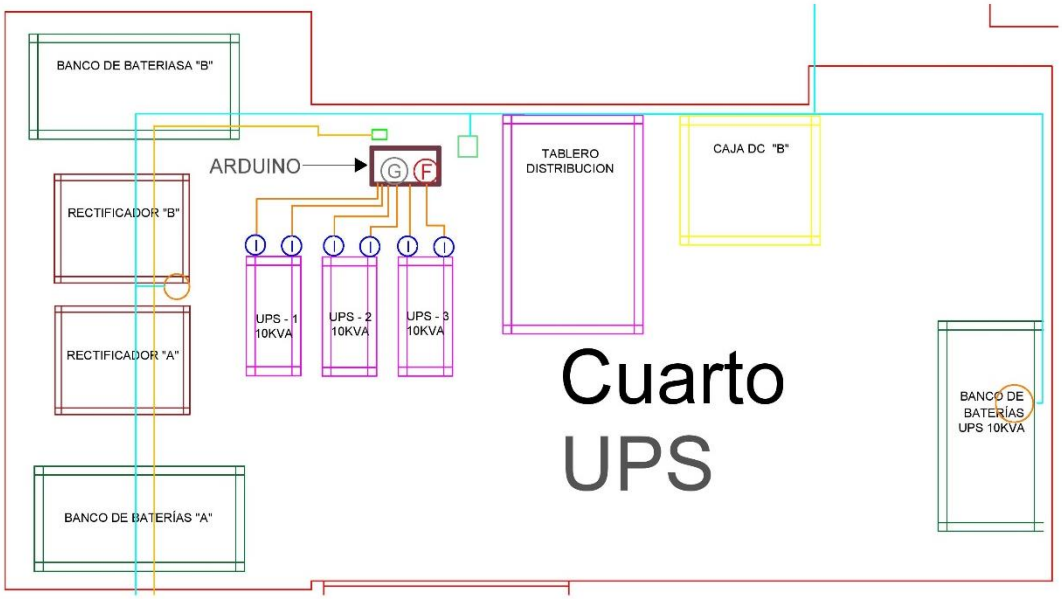


Figura 3.4: Diagrama cuarto de UPS

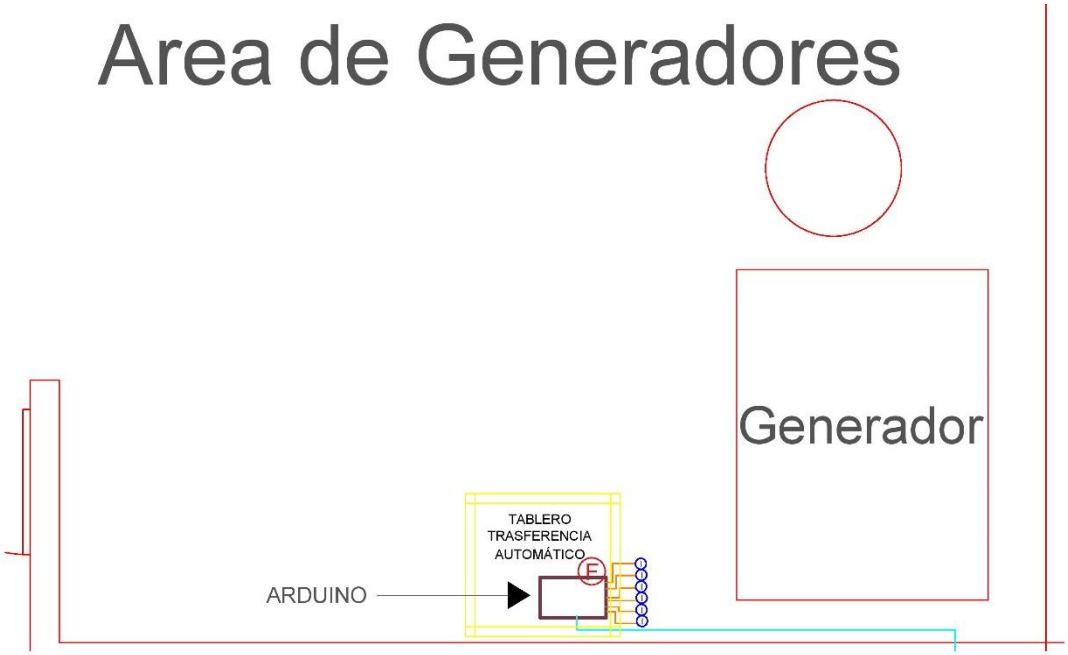


Figura 3.5: Diagrama área del generador

- **Diagrama de red**

El sistema de monitoreo está compuesto por 3 Arduinos UNO encargados de la recolección de datos de los sensores, una *Raspberry Pi* en donde están alojados la API REST, una base de datos y un *switch* que permitirá la interconexión de los elementos. Se creó una red de Arduinos que usa la topología de red fisca y lógica en estrella. A la vez, esta red está conectada a la red interna de la mini central de telecomunicaciones, como se observa en la figura 3.6.

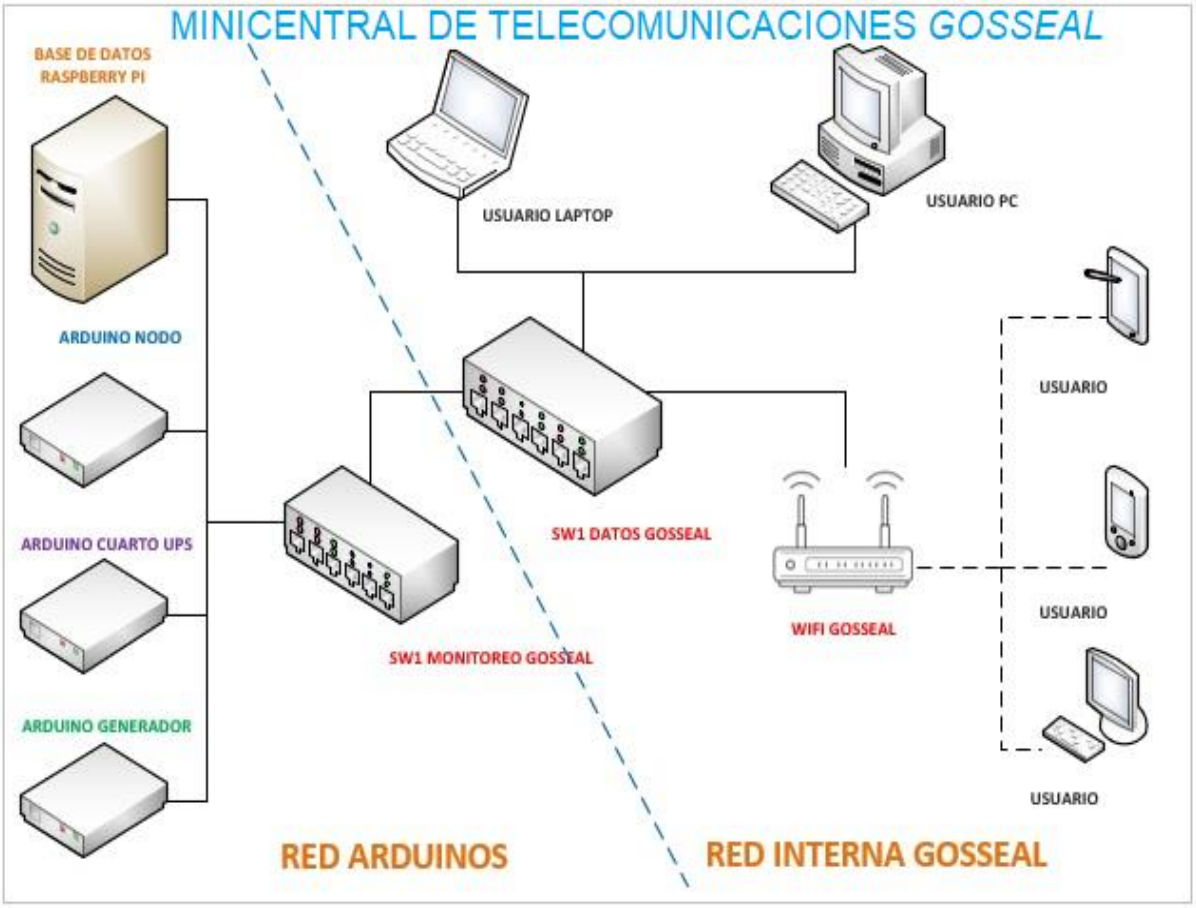


Figura 3.6: Diagrama de red nodo *Gosseal*

- **Conexiones entre Arduino y sensores**

Nodo Gosseal: es el área de la mini central de telecomunicaciones donde están alojados todos los equipos de telecomunicaciones como servidores y routers. Para esta área, con la ayuda del programa Proteus, se realizó el diseño de la figura 3.7, que contiene los sensores (temperatura y humedad DHT22, sensor de gas MQ2, sensor magnético de puerta) necesarios para monitorear las amenazas físicas. Para esta área se realizó el diseño de 5 sensores DHT de acuerdo al número de equipos importantes presentes.

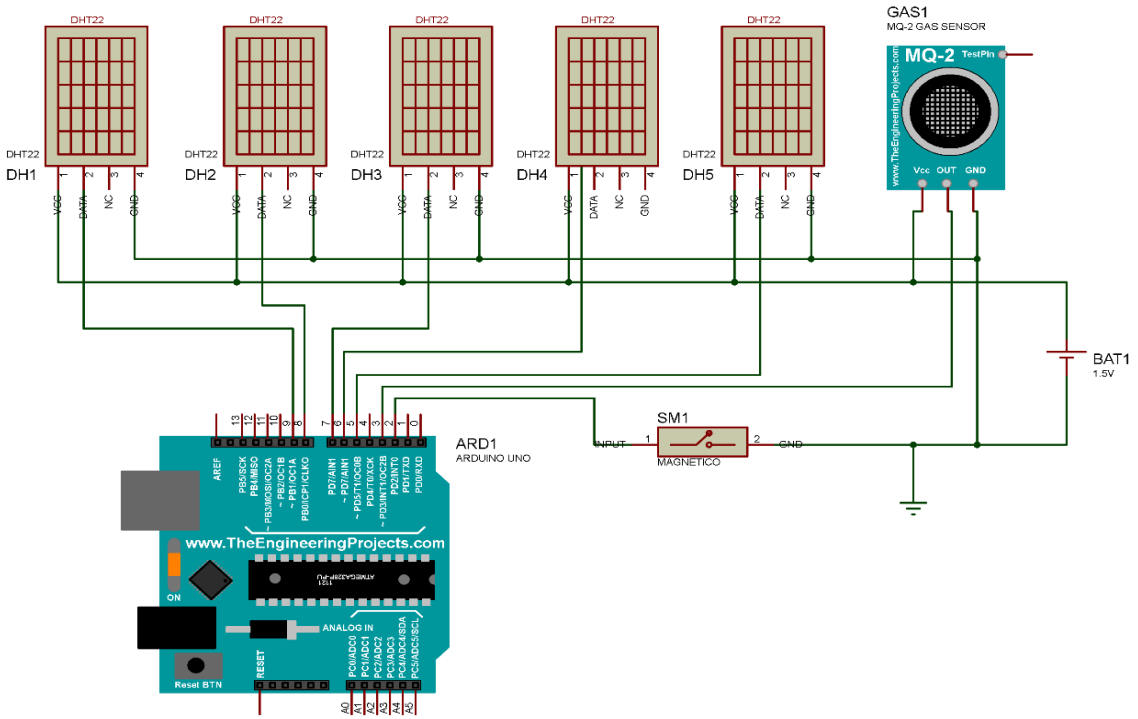


Figura 3.7: Conexión entre Arduino del nodo y sensores

Cuarto de UPS: es el área de la mini central de telecomunicaciones donde están alojados todos los equipos redundantes de energía eléctrica**.** Se realizó el diseño de la figura 3.8, que contiene 2 sensores de temperatura y humedad, 1 sensor de gas MQ2, 1 sensor de flama y 1 circuito de acondicionamiento en donde van conectados los sensores de corriente. El diagrama contiene todos los elementos necesarios para monitorear las amenazas físicas del cuarto de UPS.

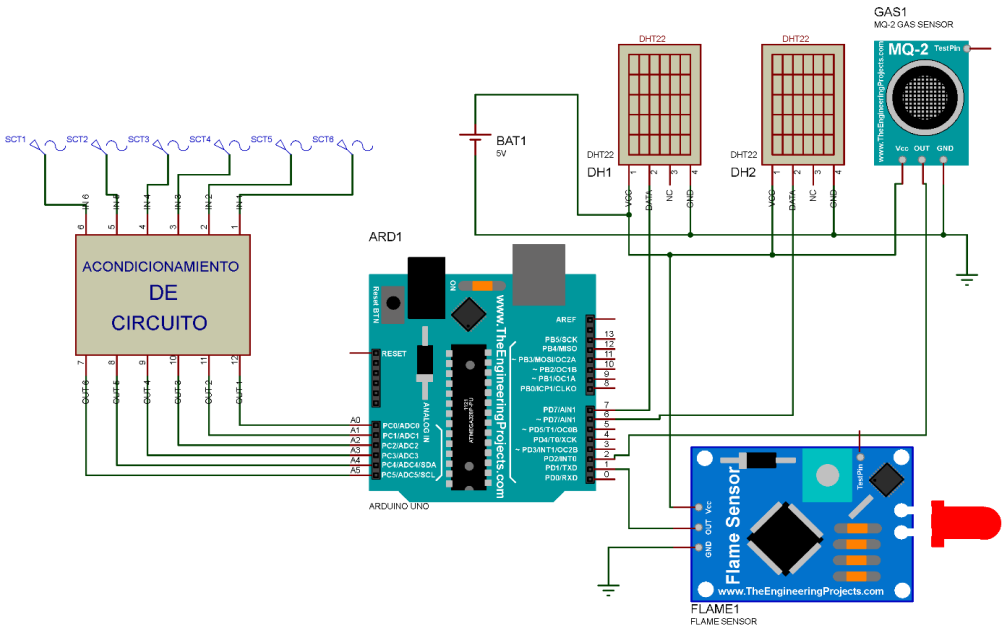


Figura 3.8: Conexión entre Arduino del cuarto de UPS y sensores

Área del generador: es el área de la mini central de telecomunicaciones donde está alojado el generador**.** Se realizó el diseño de la figura 3.9, que contiene los sensores y circuitos de acondicionamientos necesarios para monitorear las amenazas físicas. En comparación con los dos circuitos anteriores, este Arduino es el que cuenta con menos elementos conectados.

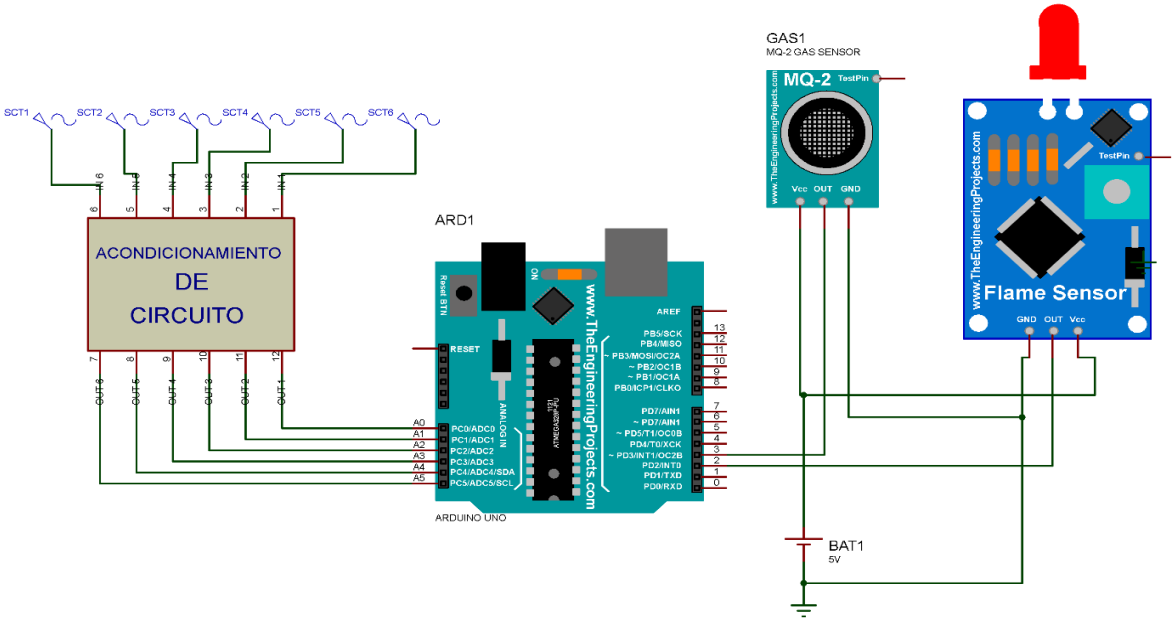


Figura 3.9: Conexión entre Arduino del generador y sensores

- **Circuitos de acondicionamiento**

Cada uno de los sensores utilizados tiene implementado internamente un circuito de acondicionamiento. Los únicos sensores que no cuentan con dicho circuito, son los sensores de corriente AC no invasivos SCT-013 y SCT-019; para los cuales se realizó dos circuitos de acoplamiento.

La salida de los sensores de corriente son ondas alternas cuyos semiciclos negativos pueden llegar a dañar el Arduino. Los pines analógicos del Arduino únicamente pueden leer voltajes entre 0 y 5 V, por lo cual el primer paso para el diseño de circuito de acondicionamiento fue añadir una resistencia de carga a la salida de los sensores. El valor de la resistencia de carga se la obtiene utilizando la Ley de Ohm (Ecuación 3.1). Para el cálculo, se debe tomar el dato de la corriente máxima de salida de los sensores, y una salida de voltaje +/- 1V.

$$V = I\ X\ R$$

$$R_{SCT-013} = \frac{1}{50\ mA} = 20\Omega \qquad R_{SCT-019} = \frac{1}{33\ mA} = 30\Omega$$

Ecuación 3.1: Ley de Ohm [57]

Luego de haber transformado la salida corriente a voltaje, se eliminó el semiciclo negativo, utilizando un rectificador de media onda. El amplificador operacional LM358 configurado como seguidor de voltaje, funciona como un rectificador de media onda. En la figura 3.10, se puede apreciar los circuitos de acondicionamiento para los sensores SCT-013 y SCT-019. La diferencia entre los 2 circuitos radica en la resistencia de carga [57].

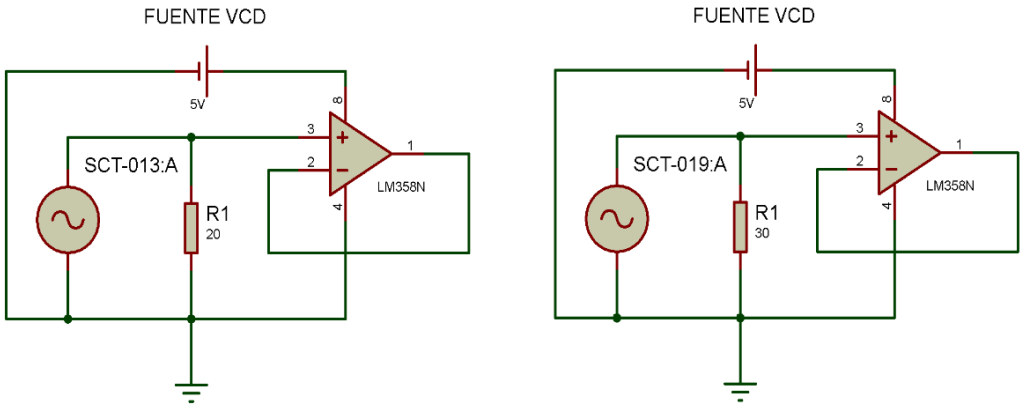


Figura 3.10: Circuito de acondicionamiento para sensores SCT

Utilizando Proteus, se simuló el funcionamiento del circuito. La señal rectificada a la salida del circuito de acondicionamiento se la puede apreciar en la figura 3.11.

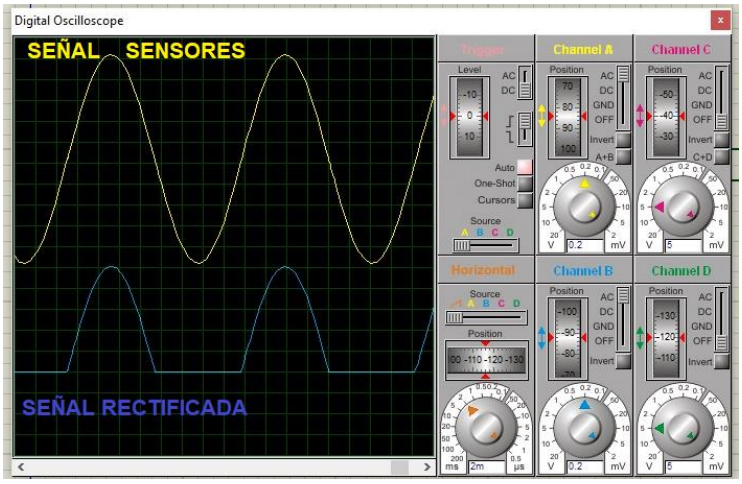


Figura 3.11: Entrada y salida de la señal de voltaje del sensor de corriente

Los circuitos antes diseñados sirven para acondicionar un solo sensor de corriente; pero ya sea para el Arduino ubicado en el cuarto de UPS o el Arduino ubicado en el área del generador, se necesita un circuito para acondicionar 6 sensores. En la figura 3.12, se puede apreciar el circuito diseñado para acondicionar 6 sensores (6 líneas de energía eléctrica). El circuito se puede utilizar para cualquier sensor SCT, solo se debe cambiar el valor de las resistencias de carga.

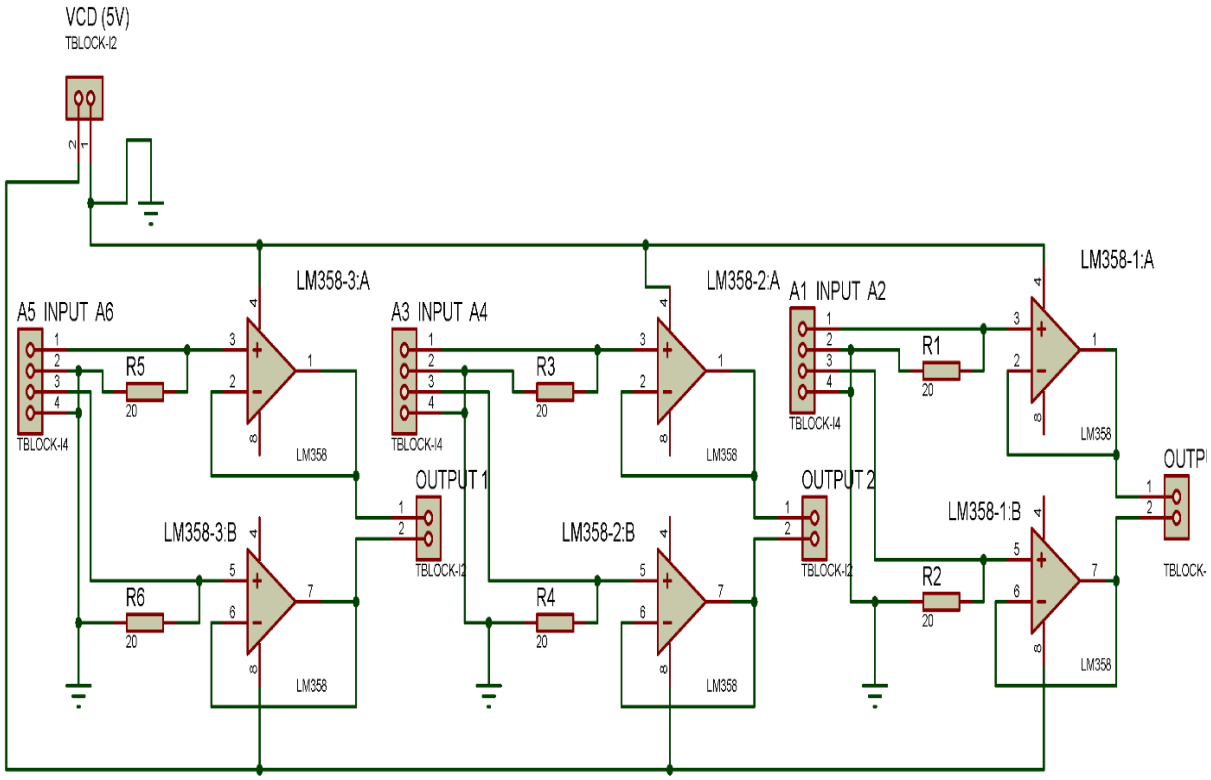


Figura 3.12: Circuito de acondicionamiento para 6 sensores

Para el diseño de la PCB (*Printed Circuit Board),* se utilizó el programa de diseño ARES de Proteus. La PCB fue diseñada en 2 capas, la PCB por la cara inferior y superior se la puede observar en la figura 3.13.

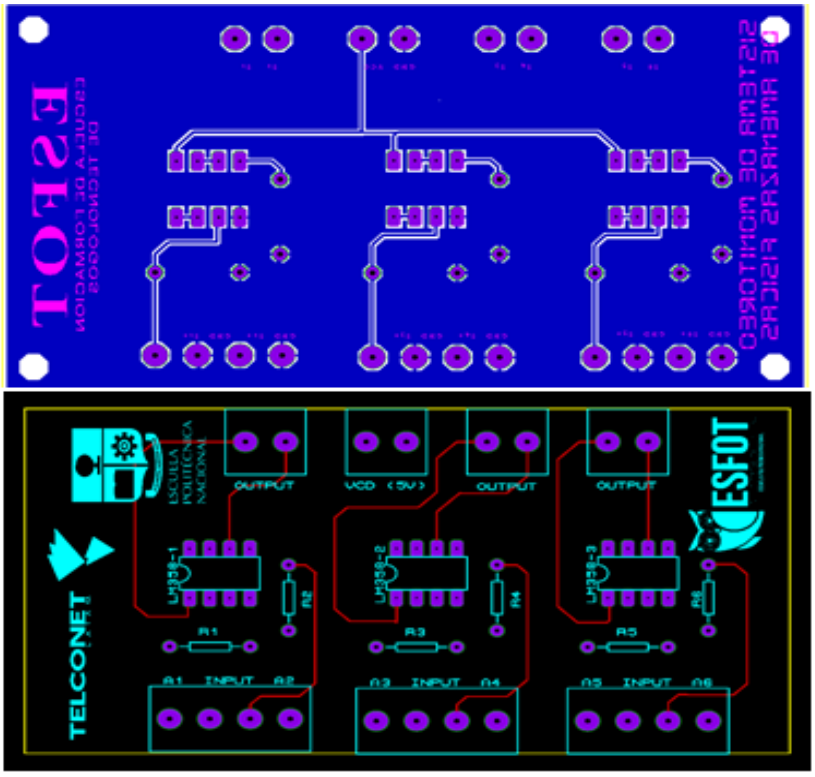


Figura 3.13: PCB cara inferior y superior

- ***Cases* para Arduinos**

Mediante el *software* en línea TinkerCAD se diseñó los *cases* o carcazas para proteger al Arduino, sensores y circuitos. Se utilizó medidas reales de los dispositivos para realizar los diseños de los tres *cases*. Los tres *cases* son diferentes porque cada Arduino va a tener diferentes sensores.

El *case* diseñado para el Arduino ubicado en el nodo (figura 3.14), cuenta con: 5 ranuras para *jacks* para conexión de los sensores de temperatura y humedad DTH22, 1 ranura circular para el sensor de gas MQ2, 2 ranuras para la conexión de la placa Arduino (serial y alimentación) y 1 ranura para el *jack* de la *shield*.

Figura 3.14: *Case* Arduino nodo

En la figura 3.15 se puede apreciar el diseño de *case* para el Arduino ubicado en el cuarto de UPS. Este *case* cuenta con: 3 ranuras para *jacks* para conexión de los sensores de temperatura y humedad DTH22, 1 ranura circular para el sensor de gas MQ2, 1 ranura semicircular para el sensor de flama, 3 ranuras rectangulares para las entradas hacia las borneras del circuito de acondicionamiento, 2 ranuras para la conexión de la placa Arduino (serial y alimentación) y 1 ranura para el *jack* de la *shield*.

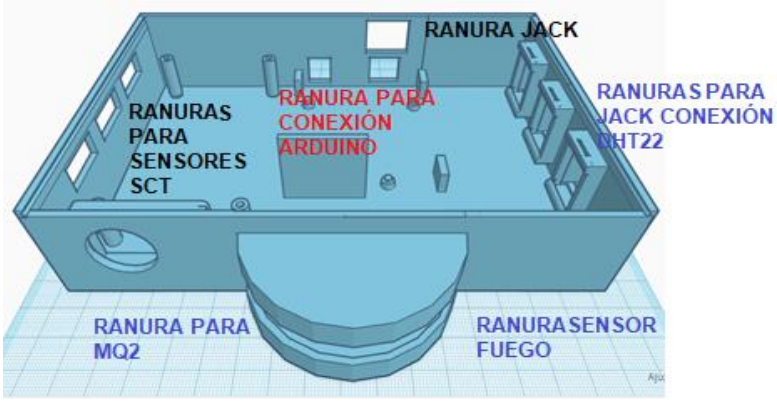


Figura 3.15: *Case* Arduino cuarto de UPS

El *case* diseñado para el Arduino ubicado en el área del generador (figura 3.16), *c*uenta con: 1 ranura para el sensor de flama, 1 ranura para el sensor de gas, 3 ranuras para las entradas hacia las borneras del circuito de acondicionamiento, 2 ranuras para la conexión de la placa Arduino y 1 ranura para el *jack* de la *shield*.

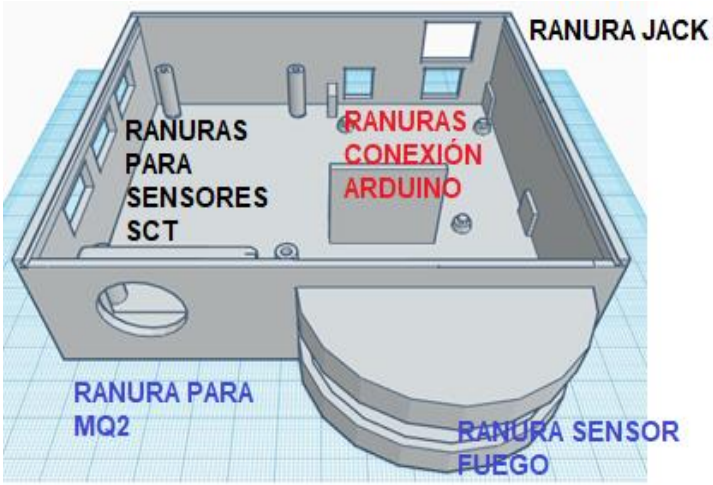


Figura 3.16: *Case* Arduino generador

Software

- **Cliente Arduino**

Bajo un Modelo Cliente-Servidor, los Arduinos se programan como clientes para que recolecten los datos de los diferentes sensores (temperatura, humedad, corriente, gas, flama) y envíen estos datos utilizando peticiones HTTP POST, a una base de datos en un servidor *web,* cada determinado tiempo. Los Arduinos son programados como cliente, por la falta de memoria para almacenar datos. También, se enviarán alertas cuando, los valores de temperatura sobrepasen los rangos establecidos, se detecte incendio (presencia de humo o flama) o fallos en líneas de red eléctrica (generador, UPSs, E.E.Q).

- **API REST**

La comunicación entre los Arduinos y la base de datos se realiza mediante una *API REST*. La API al usar REST, puede ser usada de una forma rápida y sencilla por cualquier dispositivo que utilice HTTP. La API está alojada en un servidor *web* desarrollado, utilizando un *micro framework*. El servidor *web* y *la* base de datos se instalaron en una *Raspberry Pi.* La API luego de recibir los datos, los envía a una base de datos SQL. Una página *web* con patrón de diseño MVC consulta los datos solicitados por los usuarios. Esta página *web* puede ser visualizada en un celular, *Tablet*, PC o laptop.

- **Diagrama de flujo de funcionamiento**

El sistema de monitoreo de amenazas físicas tiene como objetivo, tomar valores constantemente de los sensores del sistema, verificar que estos valores estén en rangos normales y enviarlos a la base de datos. La API es la interfaz para que se comuniquen los componentes del sistema. Este proceso se puede observar en la figura 3.17.

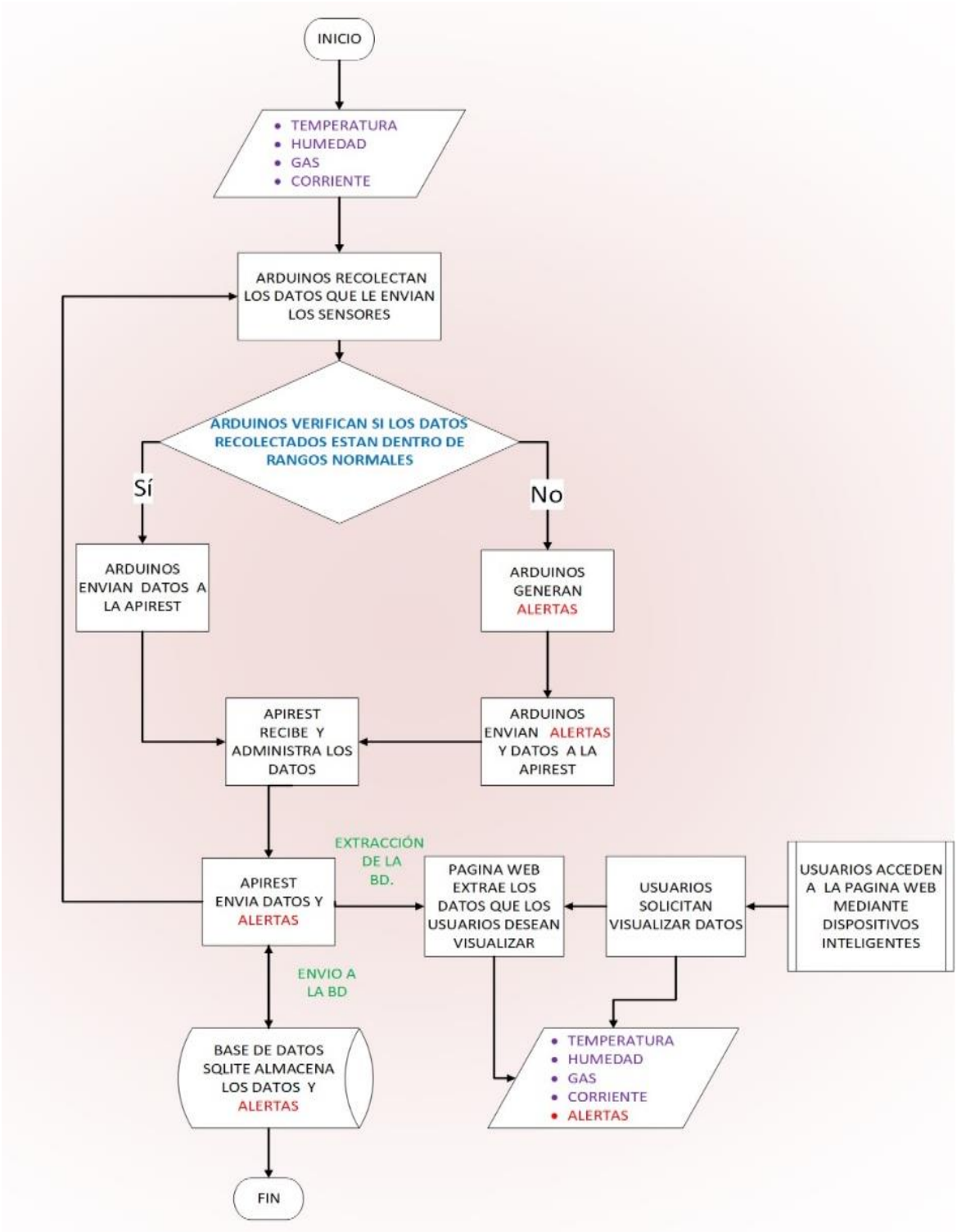


Figura 3.17: Diagrama de flujo del sistema

3.3. Implementación del sistema

Instalación de canalización

- **Canalización para el cableado de datos**

Por recomendación de Telconet, se utilizó canalización ya existente en la mini central. En los lugares que no existía, se realizó la instalación de canalización fijada a la pared, con tubo *conduit* EMT (tubería eléctrica metálica) diámetro ¾" para los tramos donde pasan hasta 4 cables UTP (capacidad 40%), como se aprecia en la figura 3.18.

Figura 3.18: Canalización

Para los tramos finales hacia los sensores, se utilizó reducciones (¾" a ½"), *Condulet* tipo LL o LR y tubos *conduit* EMT ½", como se aprecia en la figura 3.19.

Figura 3.19: Tramo final

Para la trayectoria del cableado, se utilizó uniones EMT y cajas de derivación tipo FS redondas (figura 3.20). Los tubos fueron fijados utilizando abrazaderas.

Figura 3.20: Caja de derivación tipo FS redonda

Para la salida de telecomunicaciones se utilizó un cajetín por cada Arduino. Este cajetín adicionalmente cuenta con *jacks* de conexión que sirven para conexión de los sensores DHT22 con el Arduino, como se observa en la figura 3.21.

Figura 3.21: Cajetín de datos

- **Canalización para el cableado eléctrico**

Al igual que en el cableado de datos, se utilizó canalización ya existente en la mini central. En los lugares que no contaban con la mismas, se realizó la instalación de canalización fijada

a la pared, con tubo *conduit* EMT diámetro ½". El punto final de conexión llega a un cajetín el cual tendrá una toma eléctrica polarizada, como se observa en la figura 3.22.

Figura 3.22: Punto eléctrico

Instalación de cableado de datos y eléctrico

La velocidad de transmisión requerida para el presente proyecto es inferior a 100 Mbps; motivo por el cual, se utilizó cable UTP Categoría 5e, que ofrece velocidades de hasta 1 Gbps. La cantidad de cable requerido para la transmisión de datos es de 80m incluidos remanentes y para la conexión de sensores es de 30 metros.

Para el cableado eléctrico se utilizó cable concéntrico multiflex 3x12, el cual se alimenta de un tablero de UPS para tener respaldo eléctrico.

Elaboración de *cases*

La elaboración de *cases* fue realizada por la microempresa PHOTO BOOTH. Se utilizó la técnica para fabricación de objetos conocida como impresión 3D. En la figura 3.23, se puede apreciar el *case* del Arduino que se ubicó en el área del generador.

Figura 3.23: *Case* del Arduino del generador

Elaboración de PCB

La elaboración de las PCBs para los circuitos de acondicionamiento fue realizada por la empresa internacional JLPCB, empresa líder en la elaboración de PCBs. Se tomó la decisión de realizarlo externamente y no artesanalmente porque se requería que las placas tengan acabado profesional. En la figura 3.24, se puede apreciar las placas fabricadas por JLCPCB.

Figura 3.24: PCBs terminadas

Instalación de dispositivos

El primer paso para realizar la instalación fue armar los *cases* de cada uno de los Arduinos. En cada *case* se insertaron, los diferentes dispositivos como: sensores, *jacks*, Arduino, *shield Ethernet*. Una vez insertados todos los dispositivos, se procedió a interconectarlos. En la figura 3.25, se puede apreciar uno de los *cases* terminado.

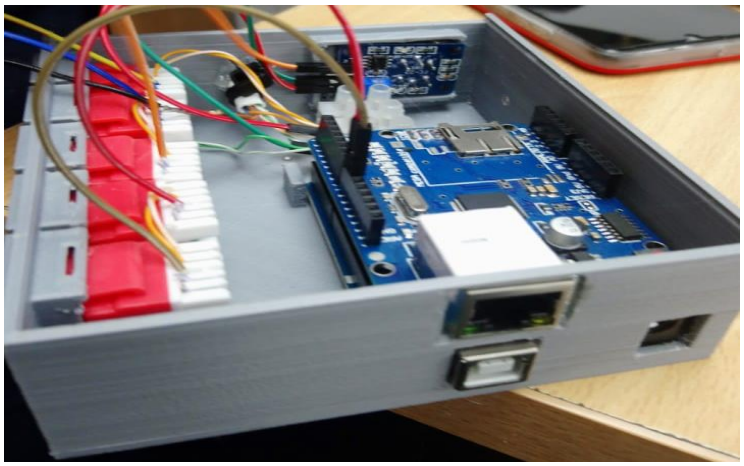


Figura 3.25: *Case* del Arduino de nodo terminado

Luego de ensamblar los *cases,* se los ubicó en sus respectivos lugares. Posterior a esto, se instaló todos los sensores en las áreas de la mini central. En la figura 3.26, se puede observar la instalación de los sensores de corriente.

Figura 3.26: Instalación de sensores SCT

Lo siguiente fue instalar el *switch* y la Raspberry Pi en el *rack* de la red interna, en la bandeja ubicada en la parte inferior; dicho lugar fue asignado por Telconet.

Finalmente, se interconectó todo el sistema utilizando *patch cords* UTP certificados. En la figura 3.27, se puede apreciar el Arduino del área del generador funcionando.

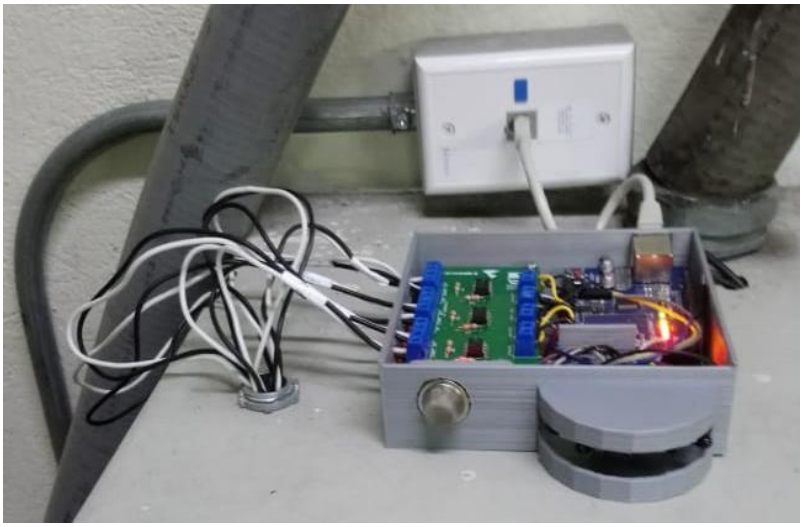


Figura 3.27: Arduino área del generador

3.4. Desarrollo del *software*

Configuración de Arduino

Todos los Arduinos tienen cargados un programa para su funcionamiento. Los programas de los 3 Arduinos son similares; la diferencia entre los programas está en los tipos de sensores conectados a ellos.

En primer lugar, se incluyó las librerías necesarias para el funcionamiento. Las librerías que permiten el funcionamiento de la *shield* son la *Ethernet* y SPI. La librería DHT permite la comunicación entre los sensores y Arduino. La librería *RestClient* permite hacer peticiones POST. La librería ArduinoJson permite el uso de JSON.

```
#include <Ethernet.h>      // LIBRERÍA PARA CONEXIÓN A INTERNET
#include <SPI.h>           // TRANSFERENCIA DE DATOS SERIE SINCRONIZADOS
#include "DHT.h"           // LIBRERÍA PARA FUNCIONAMIENTO DEL SENSOR DHT
#include <ArduinoJson.h>   // LIBRERÍA PARA TRABAJAR CON STRINGS TIPO JSON
#include "RestClient.h"    // LIBRERÍA PARA PROTOCOLO DE CLIENTE/SERVIDOR
```

En la siguiente parte del programa, se configuró las variables globales del sistema y los pines que se utilizó para conectar los sensores DHT.

```
String texto;
String response;
String login;
#define DHT1PIN 5
#define DHT2PIN 6
#define DHT1TYPE DHT22
#define DHT2TYPE DHT22
DHT dht1(DHT1PIN, DHT1TYPE);
DHT dht2(DHT2PIN, DHT2TYPE);
```

Luego, se configuró los datos del servidor para que el Arduino pueda comunicarse con este.

```
#define IP "192.168.254.240" // Server IP
#define PORT 5000     // Server Port
RestClient client = RestClient(IP, PORT);
```

En el *setup* del programa se configuró los pines como, entrada para los sensores de flama, MQ2 y magnético. También se activó la referencia interna del Arduino y se inicializó los sensores DHT. Por último, se configuró IP y MAC del Arduino, antes de inicializar la conexión a la red.

```
pinMode(2, INPUT_PULLUP);
pinMode(3, INPUT_PULLUP);
analogReference(INTERNAL);
dht1.begin();
dht2.begin();
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte ip[] = { 192, 168, 254, 242 };
```

```
Ethernet.begin(mac, ip);
```

En el bucle del programa se realizó la configuración para la lectura de sensores. Los primeros sensores que envían datos son el magnético y el MQ2 en el Arduino del nodo; adicional, los sensores de fuego y flama en los Arduinos del generador y cuarto de UPS. Si estos detectan la presencia de un evento, el programa genera y envía un mensaje de alerta. Todos los mensajes de alerta, el programa los envía a la función http_POST_log.

```
int sensorfire = digitalRead(3);
if (sensorfire == HIGH) {
texto = "Fire Alarm";
http_POST_log(texto);}

int sensorhmo = digitalRead(2);
if (sensorhmo == LOW) {
texto = "Smoke Alarm";
http_POST_log(texto); }

sensorpuerta = digitalRead(2);
if (sensorpuerta == HIGH) {
texto = "Puerta Abierta";
http_POST_log(texto);}
```

Se configuró que los siguientes sensores que envíen datos sean los DHT. El programa asigna un nombre y envía los valores recolectados a la función http POST sensor. El proceso de asignar nombres y enviar valores a la función http_POST_sensor, el programa lo hace para todos los sensores.

```
float t1 = dht1.readTemperature();
login = "T1";
http_POST_sensor(t1, "temperature", "/SensorData/temperatura");

delay(1000);
float h1 = dht1.readHumidity();
login.replace("T1", "H1");
http_POST_sensor(h1, "humidity",  "/SensorData/humedad");
```

El siguiente sensor que se configuró para enviar datos es el MQ2 en el Arduino del nodo.

```
float g1  = analogRead(A0);
login.replace("H5", "MQ1");
http_POST_sensor(g1, "cant_gas",  "/SensorData/gas");
```

En el Arduino del nodo y del generador, se configuró que los 6 SCT envíen datos. Para obtener la corriente, el programa envía el nombre del pin análogo a la función get_corriente.

Con el valor de la corriente, el programa calcula el voltaje. Este proceso se lo hizo para todos los sensores de corriente.

```
float pin = A0;
float I1 = get_corriente(pin); //Corriente eficaz (A)
login = "R-LR";
http_POST_sensor(I1, "corriente",  "/SensorData/corriente");
float V1 = I1 * 0.55 ;
http_POST_sensor(V1, "voltaje",  "/SensorData/voltaje");
```

Se configuró el programa para verificar que los valores de los sensores estén en el rango normal de funcionamiento. Si algún valor está fuera de rango o algún sensor presenta algún error en la lectura, los Arduinos generan un mensaje de alerta.

Temperatura y humedad.

```
if ( t1 >= 21 ||t2 >= 21 ||t3 >= 21 ||t4 >= 21 ||t5 >= 21)
{ texto = "Temperatura alta:Nodo"; //,  envia  mensaje
http_POST_log(texto);}

delay(1000);
if ( isnan(h1)||isnan(t2)||isnan(h3)||isnan(t4)||isnan(h5) )
{ texto = "Error lectura sensores Nodo";
http_POST_log(texto); }
```

Corriente.

```
if (I1 < 1 ||I2 < 1 ||I3 < 1  )
{ texto = "Voltaje RED OFF";
http_POST_log(texto);}
```

Gas.

```
if ( g1 > 100 )
{ texto = "Presencia Humo"; mensaje
http_POST_log(texto)}
```

Se creó 3 funciones en el código, que ayudan a realizar las acciones requeridas por el proyecto. Se desarrolló la función **get corriente** para el cálculo de la corriente. La función es de tipo *float* y es la encargada de calcular los valores de corriente. Para realizar el cálculo, la función debe recibir como parámetro el número del pin del que debe tomar los datos. Esta función internamente utiliza tres variables del tipo *float*, una del tipo *int* y una del tipo *long*. La función de tiempo millis, ejecuta el programa interno que se encarga de la toma de valores durante 500 ms, que representan 30 ciclos en una señal de 60 Hz. La variable voltajeSensor almacena los datos de un pin analógico, luego lo multiplica por la división entre el voltaje de referencia del Arduino (1.1 V) y su resolución (1023). La variable corriente calcula el valor de la corriente, multiplicando la corriente máxima de entrada del sensor SCT-013 (100 A) o del sensor SCT-019 (200 A), con el valor de la variable voltajeSensor. La variable sumatoria, calcula la sumatoria de cuadrados del valor de corriente. La variable N se incrementa hasta

q la función millis alcance los 500 ms. Luego de este proceso, el programa obtiene el valor de la variable sumatoria y el valor de N. Finalmente, duplica el valor de la sumatoria para compensar el semiciclo negativo eliminado, este valor lo divide para N y al resultado le aplica la raíz cuadrada; el valor de esta operación es la corriente Irms. Este valor es devuelto por la función. La corriente fue calculada adaptando la ecuación de RMS (Ecuación 3.2).

$$I_{RMS} = \sqrt{\frac{1}{N}\,(s1^2 + s2^2 + sn^2)}$$

Ecuación 3.2**:** I RMS [57]

```
float get_corriente(float pin) {
float voltajeSensor;
float corriente = 0;
float Sumatoria = 0;
long tiempo = millis();
int N = 0;
while (millis() - tiempo < 500) {
voltajeSensor = analogRead(pin) * (1.1 / 1024.0); //voltaje del sensor
corriente = voltajeSensor * 200.0; //corriente=VoltajeSensor*(A/1V)
Sumatoria = Sumatoria + sq(corriente); //Sumatoria de Cuadrados
N = N + 1;
delay(1); }

Sumatoria = Sumatoria * 2; //Para compensar los cuadrados de los semiciclos
corriente = sqrt((Sumatoria) / N); //ecuación del RMS
return (corriente);}
```

Se desarrolló la función **http POST sensor** para el envío de datos a la API. La función es de tipo entero y es la encargada de recibir los datos (valor del sensor, nombre del sensor, URL de la API). Esta función configura las cabeceras añadiendo los parámetros de seguridad que solicita la API (usuario: contraseña) codificado en base 64 y especificando que el contenido a enviar es JSON. Luego, crea el JSON que está compuesto por el valor del sensor y el nombre. Finalmente, con la variable del tipo entero *statusCode*, el programa envía el JSON a la URL de la API, esta variable retorna un 200 si la operación ha sido correcta.

```
int http_POST_sensor(float sensor, String tag , const char* dir_recurso) {
response = "";
client.setHeader("Authorization: Basic QWRtaW4gOkFiYzExMzM3Nw==");
client.setHeader("Content-Type: application/json");
StaticJsonBuffer<200> jsonBuffer;
char json[256];
JsonObject& root = jsonBuffer.createObject();
root[tag] = sensor;
root["login"] = login;
root.printTo(json, sizeof(json));
Serial.println(json);
```

```
int statusCode = client.post(dir_recurso, json, &response);
Serial.print(StatusCode); delay(1000);}
```

Se desarrolló la función **http POST log** para el envío de mensajes de alerta a la API. La función es de tipo entero y es la encargada de recibir los mensajes de alerta generados. La estructura de esta función es similar a la función http POST sensor. El JSON está compuesto por el mensaje de alerta.

```
float pin4 = A4;
int http_POST_log(String texto) {
String response = "";
client.setHeader("Authorization: Basic QWRtaW4gOkFiYzExMzM3Nw==");
client.setHeader("Content-Type: application/json");
const size_t bufferSize = JSON_OBJECT_SIZE(1);
DynamicJsonBuffer jsonBuffer(bufferSize);
JsonObject& root = jsonBuffer.createObject();
root["log"] = texto;
root.printTo(Serial);
char json[270];
root.printTo(json, sizeof(json));
int statusCode = client.post("/SensorData/log", json, &response);
Serial.println(statusCode);  delay(1000);}
```

Se configuró las funciones para envíos de datos y alertas en los tres Arduinos. Además de la función para el cálculo de corriente en el Arduino del cuarto de UPS y del generador.

Desarrollo de la API REST

La API se instaló en el sistema operativo *Raspbian* oficial compatible con la *Raspberry Pi.* También, se instaló el lenguaje de programación Python, lenguaje que se utilizó en el desarrollo de la API. Python usa indentación.

```
apt install python3
```

La Raspberry Pi es solo para uso exclusivo del proyecto, por lo que no hubo ningún problema en crear la carpeta del proyecto en el escritorio. Dentro de la carpeta del proyecto se instaló el entorno virtual, con la siguiente instrucción:

```
apt-get install python3-venv
```

Se activó el entorno virtual, con las siguientes instrucciones:

```
python3 -m venv venv
source venv/bin/activate
```

Se instaló los paquetes necesarios en la carpeta del proyecto, con la siguiente instrucción:

```
pip3 install flask flask-restful flask-sqlalchemy flask-marshmallow
marshmallow-sqlalchemy flask-httpauth SQLAlchemy
```

Dentro del directorio del proyecto, se creó 3 ficheros con extensión (.py) con los nombres:

- app.py
- sensor.py
- api.py

- **app.py**

Este fichero contiene los valores de configuración que se utilizan por los otros ficheros de la carpeta. El primer paso para desarrollar el fichero fue importar todos los paquetes necesarios. Luego se instanció *flask* para crear a la aplicación.

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from flask_marshmallow import Marshmallow
from flask_restful import Api
from flask_httpauth import HTTPBasicAuth
import os
app = Flask(__name__)
```

Posterior, se configuró la conexión a la base de datos.

```
app.config['SECRET_KEY'] = 'the quick brown fox jumps over the lazy dog'
app.config['SQLALCHEMY_DATABASE_URI'] = "sqlite:///db.sqlite"
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
```

Se creó dos variables. La variable db contiene la configuración de la app para SQLAlchemy, al igual que la variable ma para *Marshmallow.*

```
db = SQLAlchemy(app)
ma = Marshmallow(app)
```

Finalmente, se configuró que el fichero utilice la autenticación básica de HTTP y se creó un diccionario con usuario – clave.

```
auth = HTTPBasicAuth()
users = {"admin": "password",   "Admin" :"Abc113377"}
```

- **sensor.py**

Este fichero contiene el modelo de datos de los sensores. El primer paso para desarrollar el fichero fue importar las librerías necesarias (*date time*). También, se importó desde el fichero app las variables ma y db.

```
from app import db, ma
from datetime import datetime
```

Luego, se creó una clase de Python con el nombre, Temperatura (modelo de base de datos). La clase Temperatura, contiene como atributos un id, temperatura (valor que Arduino envíe del sensor correspondiente) y la fecha.

```
class Temperatura(db.Model):
  id = db.Column(db.Integer, primary_key=True)
  temperature = db.Column(db.Float)
  login= db.Column(db.String)
  fecha = db.Column(db.DateTime)
```

Dentro de la clase se creó un método (función dentro de la clase), el método _init_ es el constructor del objeto, este constructor inicializa los atributos de la clase y calcula la fecha

```
  def __init__(self, temperature, login, fecha=None):
    self.temperature = temperature
    self.login = login
    if fecha is None:
           fecha = datetime.now()
    self.fecha = fecha
```

Dentro del fichero se creó otra clase con el nombre *TemperaturaSchema* que utiliza la clase Temperatura, para generar un objeto que puede convertirse en JSON de forma sencilla.

```
class TemperaturaSchema(ma.ModelSchema):
  class Meta:
    model = Temperatura
```

Las instrucciones que se utilizó para los datos del sensor de temperatura, fueron las mismas para todos los sensores, el único cambio que se hizo fue el nombre de las clases. A continuación, se muestra el código que se escribió para el sensor de humedad.

```
# Clase Humedad
class Humedad(db.Model):
  id = db.Column(db.Integer, primary_key=True)
  humidity = db.Column(db.Float)
  login= db.Column(db.String)
  fecha = db.Column(db.DateTime)

  def __init__(self, humidity, login, fecha=None):
    self.humidity = humidity
    self.login = login
    if fecha is None:
           fecha = datetime.now()
    self.fecha = fecha
# Product Schema
class HumedadSchema(ma.ModelSchema):
  class Meta:
    model = Humedad
```

- **api.py**

Este fichero contiene la programación para recibir peticiones HTTP y las URL donde se publican los recursos. El primer paso para desarrollar el fichero fue importar los paquetes necesarios.

```
from flask_restful import Resource
from app import app, db, auth, users
from sensor import Temperatura, TemperaturaSchema, Humedad, HumedadSchema,
Gas, GasSchema, Corriente, CorrienteSchema, Voltaje, VoltajeSchema, Log,
LogSchema, Message, MessageSchema
import sqlalchemy
```

Luego, se creó una función que evalúa que las URLs sean seguras. Todas las funciones usan decoradores (Función que toma como entrada otra función y a su vez retorna una tercera función).

```
@auth.get_password
def get_pw(username):
    if username in users:
        return users.get(username)
    return None
```

Después se creó el objeto esquema para el sensor de temperatura, este está disponible en todo el fichero. Luego se definió la URL para publicar los recursos y un decorador para especificar que la función necesita autentificación.

```
temperatura_schema = TemperaturaSchema()
@app.route('/SensorData/temperatura', methods=['POST'])
@auth.login_required
```

Posterior, se creó la función *add_temperatura* que utilizó el método POST para que la API reciba datos del Arduino; estos datos (valor del sensor y nombre) llegaron en JSON, el programa llamó a la clase Sensor, la cual armó el objeto para guardarlo en la base de datos con *db.session.add.* Si la operación fue correcta, el programa retorna al Arduino un mensaje y el código 200.

```
def add_temperatura():
  args = request.get_json(force=True)
  sensor_read = Temperatura(args["temperature"], args["login"])
  db.session.add(sensor_read)
  db.session.commit()
  return 'Temperatura Registrada', 200
```

Se creó la función get_temperaturas que utiliza el método GET para obtener los datos desde la base de datos. Al igual que en la función anterior, se definió una URL. La información es consultada con la instrucción *query* y all indica al programa que se necesita toda la información de la clase Temperatura. Con la función *jsonify* la información es devuelta en JSON.

```
@app.route('/SensorData/temperatura', methods=['GET'])
def get_temperaturas():
  all_temperaturas = Temperatura.query.all()
  return temperaturas_schema.jsonify(all_temperaturas)
```

Posterior, se creó la función get_temperatura, que utiliza el método GET, para obtener los datos de la base de datos. Al igual que en las dos funciones anteriores, se definió una URL. A esta función se le añade el parámetro *login*, que sirve para que se pueda obtener información filtrada de la clase Temperatura. Con la función *jsonify* la información es devuelta en JSON.

```
@app.route('/SensorData/temperatura/<string:login>', methods=['GET'])
#@auth.login_required
def get_temperatura(login):
  all_temperaturas = Temperatura.query.filter_by(login=login )
  seq = [[item.date, item.temperature] for item in all_temperaturas]
  return jsonify(seq)
```

Las instrucciones que se utilizó para enviar u obtener información de la base de datos del sensor de temperatura, fueron las mismas para todos los sensores. Las modificaciones que se realizaron fueron las siguientes: cambiar el nombre de las funciones, cambiar las URLs de la API y crear objetos con los diferentes esquemas importados del fichero sensor.py. A continuación, se muestra el código que se escribió para los sensores de humedad.

```
humedad_schema = HumedadSchema()
humedades_schema = HumedadSchema(many=True)
# Ingresar humedades
@app.route('/SensorData/humedad', methods=['POST'])
@auth.login_required
def add_humedad():
  args = request.get_json(force=True)
  sensor_read = Humedad(args["humidity"], args["login"])
  db.session.add(sensor_read)
  db.session.commit()
  return 'Humedad Registrada', 200
@app.route('/SensorData/humedad', methods=['GET'])
@auth.login_required
def get_humedades():
  all_humedades = Humedad.query.all()
  return humedades_schema.jsonify(all_humedades)
@app.route('/SensorData/humedad/<string:login>', methods=['GET'])
```

```
@auth.login_required
def get_humedad(login):
  all_humedades = Humedad.query.filter_by(login=login )
  seq = [[item.date, item.humidity] for item in all_humedades]
  return jsonify(seq)
```

Finalmente, se creó las líneas de código que ponen a funcionar la API. Con el parámetro host se estableció que el servidor pueda ser accesible desde cualquier computadora de la redd. Con el parámetro debug, como verdadero, se estableció que se actualice el servidor cada que se realice un cambio en el código (fase de desarrollo).

```
if __name__ == '__main__':
  app.run(host="0.0.0.0",debug=True)
```

Base de Datos

Se creó la bd utilizando los ficheros sensor y app. Se utilizó una consola de Python para importar desde el fichero app.py la variable db.

```
from app import db
```

Se importó desde el fichero sensor.py las clases creadas (modelos de datos de SQLAlchemy).

```
from sensor import Temperatura
```

Por último, con la línea de comando db.create_all, se generó la base de datos sqlite.

```
from app import db
from sensor import Temperatura
from sensor import Humedad
from sensor import Gas
from sensor import Corriente
from sensor import Voltaje
from sensor import Log
db.create_all()
```

En el directorio se creó automáticamente el fichero db.sqlite. La estructura de la base de datos (figura 3.28), está conformada por 6 tablas, estas a la vez contienen las columnas con el nombre de los datos a almacenar. Se utilizó el programa SQLITE BROWSER para realizar esta visualización.

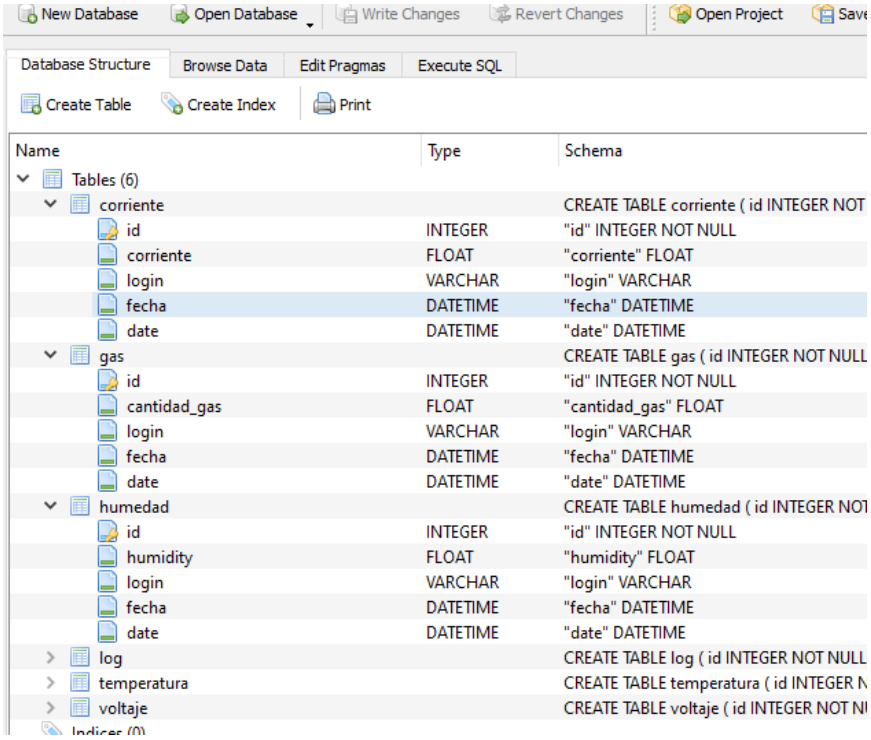


Figura 3.28: Estrcutura de la base de datos

Las tablas de la base de datos están conformadas por columnas y filas, similar a una hoja de cálculo, como se puede apreciar en la figura 3.29. Las filas son los registros de los datos de un sensor en específico. Las columnas contienen el tipo de información que aparece en los registros.

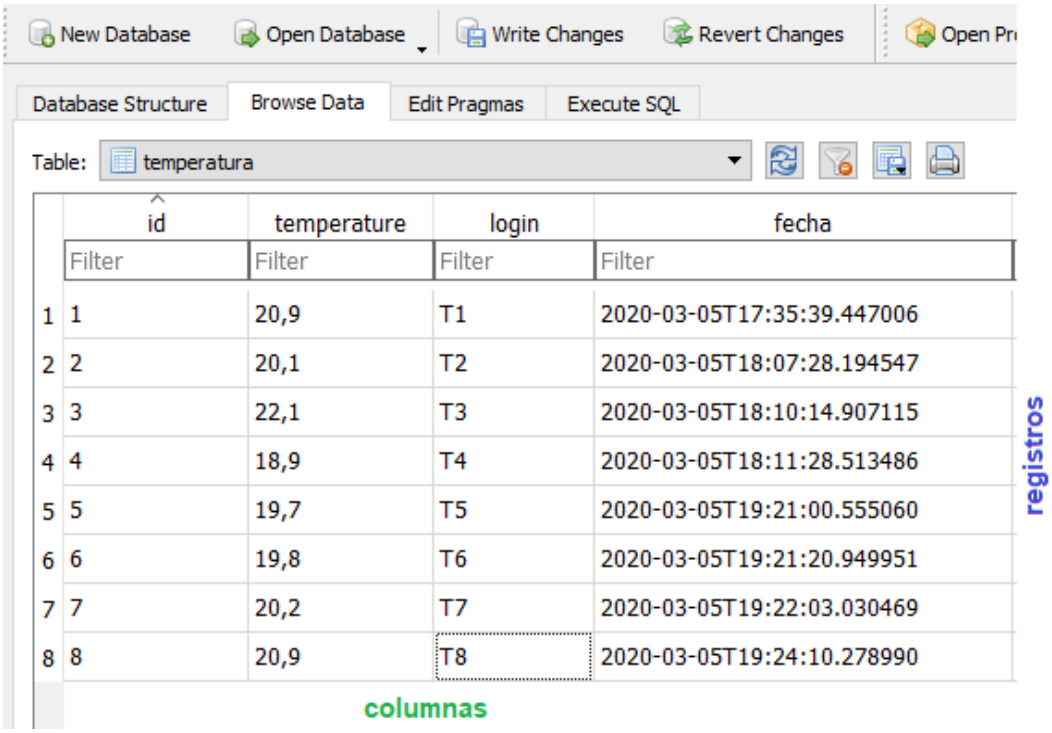


Figura 3.29: Tabla de valores de sensores de Temperatura

La tabla 3.1, se exportó desde la base de datos. Esta tabla contiene los valores de los sensores de temperatura. La primera columna genera automáticamente un identificador cuando se ingrese un registro. La segunda columna almacena los valores de temperatura de todos los sensores DHT. La tercera columna almacena el nombre sensor. La cuarta columna genera la fecha automáticamente cuando se ingrese el registro. Todas las tablas tienen la misma estructura.

Tabla 3.1: Tabla temperatura de la base de datos

ID	TEMPERATURA	*LOGIN*	FECHA
1	20,9	H1	2020-03-05T17:35:39.447006
2	20,1	H2	2020-03-05T18:07:28.194547
3	22,1	H3	2020-03-05T18:10:14.907115
4	18,9	H4	2020-03-05T18:11:28.513486
5	19,7	H5	2020-03-05T19:21:00.555060
6	19,8	H6	2020-03-05T19:21:20.949951
7	20,2	H7	2020-03-05T19:22:03.030469
8	20,9	H1	2020-03-05T19:24:10.278990

Desarrollo de la página *web*

Adicional a los ficheros creados anteriormente, dentro del directorio del proyecto se creó 2 carpetas adicionales:

- *static*
- *templates*

La carpeta *static,* contiene todas las librerías necesarias para mostrar las vistas del sitio *web*. Todas las vistas están dentro de la carpeta *templates*.

- **Modelo**

Se utilizó el fichero sensor.py como el modelo del patrón mvc.

- **Controlador**

El fichero api.py se utilizó como controlador. Para esto, se estableció todas las rutas URL de la página *web* dentro de este fichero. Se configuró que todas las rutas tengan decoradores.

La primera ruta que se creó fue la de *home* (*/*); esta ruta utiliza los métodos GET Y POST, para verificar que el *password ingresado* en el formulario sea correcto, asignarle una sesión y redirigir al usuario a la ruta *index.*

```
@app.route('/', methods = ['GET', 'POST'])

def logeo():
  if request.method == 'POST':
    session.pop('user', None)

    if request.form['password']== 'xxxxxxx':
      session['user'] = request.form['username']
      return redirect(url_for("index"))
  return render_template('login.html')
```

La siguiente ruta que se creó fue *index*, esta ruta retorna la vista del mismo nombre. Todas las rutas retornan vistas utilizando la función *render template*.

```
@app.route('/index')
def index():
  if g.user:
    return render_template('index.html', user=session['user'])
```

La siguiente ruta que se creó fue /salir, que permite al usuario salir de la página *web*, redirigiendo al usuario a la página de *login* y cerrando la sesión.

```
@app.route('/salir')
def salir():
  session.pop('user', None)
  return redirect(url_for('logeo'))
```

A continuación, se creó rutas para mostrar en la página *web* los valores de todos los sensores en tablas. Se creó la ruta /temp que devuelve la vista del mismo nombre. Esta ruta muestra en la página web los datos de todos los sensores de temperatura en una tabla.

```
@app.route('/temp')
def temp():
  if g.user:
    return render_template('temp.html', user=session['user'])
```

Las instrucciones creadas para mostrar los valores de los sensores de temperatura en una tabla, es la misma para todos los sensores. Las modificaciones que se realizaron fueron el cambio de nombre de las funciones y las vistas a retornar. A continuación, se muestra el código que se escribió para mostrar los valores de humedad en una tabla.

```
@app.route('/hum')
def hum():
  if g.user:
    return render_template('hum.html', user=session['user'])
```

Seguidamente, se creó rutas para mostrar en la página *web* los valores de todos los sensores en gráficos en tiempo real. Se creó la ruta /chart que devuelve la vista del mismo nombre. Esta ruta muestra en la página *web* los datos de todos los sensores de temperatura en gráficos individuales.

```
@app.route('/chart')
def chart():
  if g.user:
    return render_template('chart.html', user=session['user'])
```

Las instrucciones creadas para mostrar los valores de los sensores de temperatura en gráficos individuales, es la misma para todos los sensores. Las modificaciones que se realizaron fueron el cambio de nombre de las funciones y las vistas a retornar. A continuación, se muestra el código que se escribió para mostrar los valores de humedad en gráficos.

```
@app.route('/charh')
def charh():
  if g.user:
    return render_template('charh.html', user=session['user'])
  return redirect(url_for('logeo'))
```

- **Vista**

Se creó las vistas en carpeta *templates* del directorio. En esta carpeta se encuentran la vista para que el usuario se autentique, vista de la página principal, vistas para mostrar los datos en tablas y vistas para mostrar los datos en forma de gráficos, como se muestra en la figura 3.30.

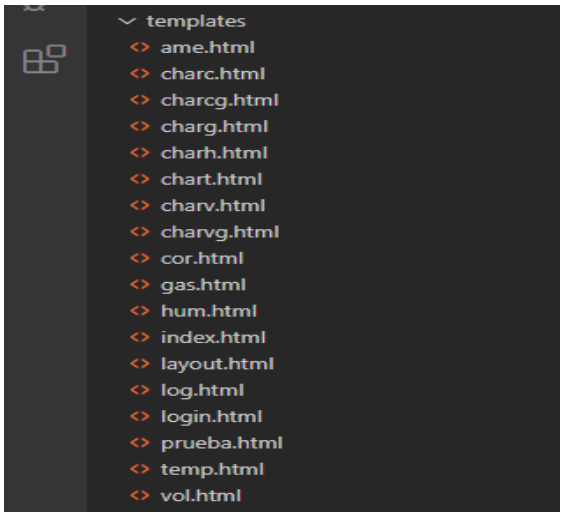


Figura 3.30: Vistas del modelo MVC del proyecto

Login.html: esta vista se creó con el objetivo de proteger el acceso a cualquier ruta de la página *web*. Se dividió la página en una cuadrícula de 8 columnas de las 12 en total, se las

alineó al centro. A esta cuadrícula se le aplica un color de fondo blanco y se la dividió en 3 partes con las etiquetas <*div*>. En la primera sección se insertó la imagen de Telconet con atributos y un mensaje de bienvenida. Para las otras divisiones se utilizó la clase "*field*", también se les asignó atributos como ícono y *placeholde*r, que indica al usuario qué dato ingresar. Para cada división se utilizó la etiqueta <*input*> de tipo texto para el nombre y de tipo *password* para la contraseña. Finalmente, se creó un botón para enviar la información al controlador utilizando el método POST.

```
<div class="row">
  <div class="col-xs-8">
     <div class="columns is-centered">
        <div class="column is-half"style="background-color:white;">
           <div align="center"><img src="img/telco.jpg" width="350"
            height="2"></div>
           <p class="login-box-msg" style="color:blue" align="center">texto
            </p>

            <form method="POST">
              <div class="field">
                <label class="label">Usuario</label>
                <div class="control has-icons-left has-icons-right">
                  <input class="input" type="text" name="username"
                    placeholder="Username">
                  <span class="icon is-small is-left">
                    <i class="fas fa-user"></i>
                  </span>
                </div>
              </div>
              <div class="field">
                <label class="label">Contraseña</label>
                <p class="control has-icons-left">
                   <input class="input" type="password" name="password"
                     placeholder="Password">
                   <span class="icon is-small is-left">
                        <i class="fas fa-lock"></i>
                   </span>
                </p>
              </div>
              <div class="field">
                 <p class="control">
                     <button type="submit" class="button is-success">
                        Login
                     </button>
                 </p>
              </div>
            </form>
        </div>
     </div>
  </div>
</div>
```

index.html: esta vista se creó con el objetivo de mostrar una página de bienvenida luego que el usuario se autentique. Todas las vistas del proyecto utilizan la plantilla l*ayout.html.* Para usar la plantilla en cada vista se escribió tres líneas de código.

```
{% extends "layout.html" %}

{% block content %}

LINEAS DE CODIGO

{% endblock %}
```

Esta vista es la página principal del proyecto, en esta se insertó 2 imágenes que contiene información acerca de Telconet

```
{% extends "layout.html" %}

{% block content %}
<img src="static/dist/img/telco5.jpg" >
<img src="static/dist/img/telco2.jpg" >
{% endblock %}
```

Vistas de tablas de datos: esta vista se creó con el objetivo de mostrar los valores de todos los sensores que sean de un mismo tipo en una sola tabla. Se creó la tabla con la etiqueta <*table*>, en esta etiqueta se especificó la altura, el estilo de encabezado y se configuró la tabla como auto recargable. También se estableció el orden para mostrar los datos, se configuró la URL de la API de donde se obtendrá los datos y la opción para exportar datos.

```
<table id="table" data-height="750" data-header-style="headerStyle" data-
show-refresh="true" data-auto-refresh="true"
  data-url="http://192.168.100.48:5000/SensorData/temperatura" data-
search="true" data-strict-search="true"
  data-pagination="true" data-sort-name="name" data-sort-order="desc" data-
show-toggle="true" data-show-columns="true"
  data-show-export="true" data-export-data-type="all">
```

Con la etiqueta <*thead*>, se configuró las filas que representan el encabezado de la tabla. Con la etiqueta <tr>, se estableció las filas de la tabla. Con la etiqueta <th>, se configuró el encabezado de la tabla.

```
<thead>
    <tr>
      <th data-field="id" data-sortable="true"># ID</th>
      <th data-field="login" data-sortable="true">Sensor</th>
      <th data-field="temperature" data-sortable="true">Temperatura</th>
      <th data-field="fecha" data-sortable="true">Fecha</th>
    </tr>
  </thead>
</table>
```

Finalmente, se creó un *script* que sirve para establecer el estilo del encabezado.

```
<script>
  var $table = $('#table')
  $(function () {
    $table.bootstrapTable()
  })
  function headerStyle(column) {
    return {
      id: {css: { background: 'lavender', color: 'midnightblue'}  },
      login: {css: { background: 'lavender', color: 'midnightblue' }  },
      temperature:{css: { background: 'lavender', color: 'midnightblue'} },
      fecha: {css: { background: 'lavender', color: 'midnightblue' }   }
           }[column.field]}
</script>
```

Las instrucciones creadas para mostrar los valores de todos los sensores de temperatura en una tabla, es la misma para todos los sensores. Las modificaciones que se realizaron fueron cambiar la URL, cambiar de nombre de las funciones y las vistas a retornar. A continuación, se muestra el código que se escribió para los valores de los sensores de humedad.

```
{% extends "layout.html" %}

{% block content %}
<table id="table" data-height="750" data-header-style="headerStyle" data-
show-refresh="true" data-auto-refresh="true"
  data-url="http://192.168.100.48:5000/SensorData/humedad" data-
search="true" data-strict-search="true"
  data-pagination="true" data-sort-name="name" data-sort-order="desc" data-
show-toggle="true" data-show-columns="true"
  data-show-export="true" data-export-data-type="all">

    <tr>
      <th data-field="id"data-sortable="true"># ID</th>
      <th data-field="login"data-sortable="true">Sensor</th>
      <th data-field="humidity"data-sortable="true">Humedad</th>
      <th data-field="fecha"data-sortable="true">Fecha</th>
    </tr>
  </thead>
</table>

<script>
  var $table = $('#table')
  $(function() {
    $table.bootstrapTable()
  })
     function headerStyle(column) {
      return {
      id: {css: { background: 'lavender', color: 'midnightblue'}  },
      login: {css: { background: 'lavender', color: 'midnightblue' }  },
      temperature:{css: { background: 'lavender', color: 'midnightblue'} },
      fecha: {css: { background: 'lavender', color: 'midnightblue' }   }
           }[column.field]}
</script>
```

Vistas de gráficos en tiempo real: esta vista se creó con el objetivo de mostrar los valores de todos los sensores de un mismo tipo mediante gráficos individuales en una sola página. Se creó los gráficos utilizando las plantillas de HIGHCHARTS. Se dividió la página en dos secciones de seis columnas.

```
<div class="row">
 <div class="col-md-6">
```

La sección de la izquierda se configuró para mostrar los gráficos de los sensores de corriente de las líneas de UPS1, UPS2 y UPS3. Con este fin, se dividió la sección en tres contenedores. Al primer contenedor se le asignó un id y las dimensiones del gráfico.

```
<div class="card-body">
   <div id="container1" style="height: 400px; min-width: 310px; max-width:
      600px"></div>
```

Se creó un *script* para generar el gráfico, este se ejecuta con los atributos configurados previamente. En este *script* se definió el tipo de gráfico, se le asignó una descripción y se estableció la URL de la API de donde se obtendrá los datos. Se estableció que la primera fila del conjunto de datos no se tome como nombre de la serie. Finalmente, con *enablePolling* se configuró que el gráfico se actualice de forma automática.

```
<script>
  Highcharts.chart('container1', {
     chart: { type: 'spline'},
     title: { text: 'Corriente UPS1'},
     subtitle: {text: 'Linea: 1'},
     data: {
         rowsURL: 'http://192.168.100.70:5000/SensorData/corriente/U1-L1',
         firstRowAsNames: false,
         enablePolling: true}
         });
</script>
```

Las instrucciones creadas para mostrar el gráfico de un sensor de corriente, son las mismas que para el resto de sensores. Las modificaciones que se realizaron fueron la URL, el id y la descripción de gráfico. A continuación, se muestra el código que se escribió para mostrar un valor de voltaje.

```
<div class="card-body">
  <div id="container3" style="height: 400px; min-width: 310px; max-width:
    600px"></div>
<script>
   Highcharts.chart('container3', {
       chart: {type: 'spline'},
       title: { text: 'Corriente UPS2'},
     subtitle: {text: 'Linea: 1'},
     data: {
         rowsURL: 'http://192.168.100.70:5000/SensorData/voltaje/U1-L2',
         firstRowAsNames: false,
         enablePolling: true}
```

```
});</script>
```

Todo el proceso anterior que describe la creación de tres tablas en el lado derecho, se lo hizo también para los gráficos en el lado izquierdo.

***Layout*:** es una plantilla que contiene todos los elementos de la página *web*. Esta plantilla se utilizó para simplificar el desarrollo de todas las vistas. Este fichero se creó utilizando la plantilla *blank page* de *Admin* LTE (figura 3.31).

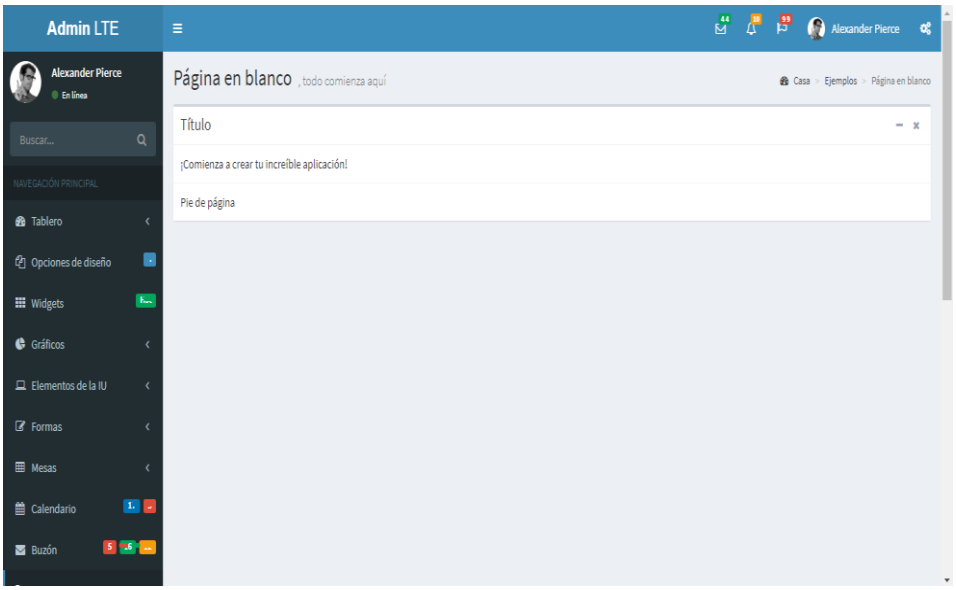


Figura 3.31: Plantilla *blank page Admin* LTE

Se descargó las librerías en la carpeta *static* y el archivo *blank page* en *templates*. Se cambió el nombre *de blank page* por *layout.html*. Este fichero se editó y adaptó a las necesidades de Telconet.

Lo primero que se hizo al editar el fichero, fue importar los archivos necesarios para el funcionamiento. Estos archivos son hojas de estilo y librerías.

```
<link rel="stylesheet" href="static/tables/bootstrap-table.min.css" >
<link rel="stylesheet"
href="static/bower_components/bootstrap/dist/css/bootstrap.min.css">
<link rel="stylesheet" href="static/bower_components/font-awesome/css/font-
awesome.min.css">
<link rel="stylesheet"
href="static/bower_components/Ionicons/css/ionicons.min.css">
<link rel="stylesheet" href="static/dist/css/AdminLTE.min.css">
<link rel="stylesheet" href="static/dist/css/skins/_all-skins.min.css">
<script src = "static/dist/js/jquery.min.js"></script>
<script src = "static/dist/js/moment.min.js"></script>
```

El *main-header* está ubicado en la parte superior y contiene en la parte izquierda el logo de Telconet y un mini logo cuando se minimiza la barra.

```
<a href="/index" class="logo">      <
<span class="logo-mini"><b>TN</b></span>
<span class="logo-lg"><b>TELCO </b>MONITOREO</span>
</a>
```

El *content-wrapper o contenido* principal, contiene la información mostrada en tablas o gráficos. En esta sección se creó una tabla en la parte superior que muestra las alertas del sistema. Al ser una aplicación que se desarrolló para uso exclusivo de Telconet, se agregó un pie de página indicando este detalle. Con la etiqueta se hizo una división, con la clase *box-footer* se indicó que era un pie de página. En la siguiente línea de código se añadió texto, con negrilla.

```
<div class="box-footer">
<strong>Copyright &copy; 2018 <a href="https://www.telconet.net/">Telconet
S.A</a>.</strong> | Todos los Derechos Reservados.
</div>
```

El *sidebar-wrapper* o barra lateral, contiene menús de selección de información. En esta sección se creó dos menús desplegables. El primer menú muestra el tipo de información que el usuario puede ver en una tabla. Con la etiqueta <li> se crea una lista, con la clase *treeview* se crea una lista tipo árbol. Con la etiqueta <i> se asignó un ícono, mientras que con la etiqueta <*span*> se añadió texto.

```
<li class="treeview">
<a href="#">
<i class="fa fa-database"></i>
<span>Tablas de Datos </span>
<span class="pull-right-container">
<i class="fa fa-angle-left pull-right"></i>
</span>
</a>
```

Para asignar los elementos del menú desplegable, se escribió las siguientes líneas de código. Con la etiqueta <ul> y la clase *treeview* se creó una lista desordenada. Se utilizó la etiqueta <li> para crear los elementos del menú, a estos elementos se les asignó un nombre, un ícono y se les asignó los enlaces correspondientes.

```
<ul class="treeview-menu">
<li><a href="temp"><i class="fa fa-thermometer-empty"></i>
Temperatura</a></li>
<li><a href="hum"><i class="fa fa-snowflake-o"></i> Humedad</a></li>
<li><a href="cor"><i class="fa fa-bolt"></i> Corriente</a></li>
<li><a href="vol"><i class="fa fa-battery-half"></i> Voltaje</a></li>
```

```
<li><a href="gas"><i class="fa fa-fire"></i> MQ2</a></li>
<li><a href="log"><i class="fa fa-pencil-square-o"></i> Logs</a></li>
</ul>
```

El segundo menú muestra el tipo de información que el usuario puede ver en gráficos. El proceso para escribir esta parte del código fue el mismo que en la sección de tablas.

```
<li class="treeview">
 <a href="#">
   <i class="fa fa-play-circle-o"></i>
   <span>Tiempo real </span>
   <span class="pull-right-container">
   <i class="fa fa-angle-left pull-right"></i>
   </span></a>
 <ul class="treeview-menu">
  <li><a href="chart"><i class="fa fa-thermometer-empty"></i>Temp</a></li>
  <li><a href="charh"><i class="fa fa-snowflake-o"></i> Humedad</a></li>
  <li><a href="charc"><i class="fa fa-bolt"></i> Corriente UPS</a></li>
  <li><a href="charcg"><i class="fa fa-bolt"></i> Corriente gen</a></li>
  <li><a href="charv"><i class="fa fa-vimeo"></i> Voltaje UPS</a></li>
  <li><a href="charvg"><i class="fa fa-vimeo-square"></i> Voltaje </a></li>
  <li><a href="charg"><i class="fa fa-fire"></i> MQ2</a></li>
  <li><a href="log"><i class="fa fa-pencil-square-o"></i> Logs</a></li>
 </ul>
</li>
```

Se escribió las siguientes líneas con el objetivo de que la plantilla sea utilizada en cualquier vista.

```
{% block content %}

{% endblock %}
```

3.5. Pruebas de funcionamiento

Pruebas de encendido del sistema

Se verificó visualmente que los componentes del sistema estén encendidos, luego de haberlos energizado. Se energizó los Arduinos con las fuentes de poder, vía USB. Los sensores DHT, los circuitos de acondicionamiento y los sensores de flama fueron energizados por una fuente externa. Las dos fuentes de poder fueron conectadas a las tomas eléctricas del sistema. Además, el *switch* y la Raspberry Pi se energizaron con sus propias fuentes de poder. Estos dispositivos fueron conectados a las tomas eléctricas del *rack* de la red interna de la mini central.

Pruebas de sensores

Se realizó lecturas de cada uno de los sensores instalados a través del monitor serie, herramienta de Arduino, donde se fue corroborando el correcto funcionamiento de cada uno.

- **Sensor DHT 22**

El DHT22 tiene su propio sistema de comunicación bidireccional mediante un único conductor empleando señales temporizadas. En cada envío de valores, el sensor trasmite un total de 40 bits, en 4ms. Estos 40 bits corresponden a 2 *bytes* para la medición de humedad, 2 *bytes* para la medición de temperatura y 1 *byte* final para la comprobación de errores. Con estos datos se procedió a realizar pruebas de medición del sensor (figura 3.32). En la tabla 3.2, se muestra algunos valores medidos por un sensor DHT y los valores medidos por el termómetro de la mini central.

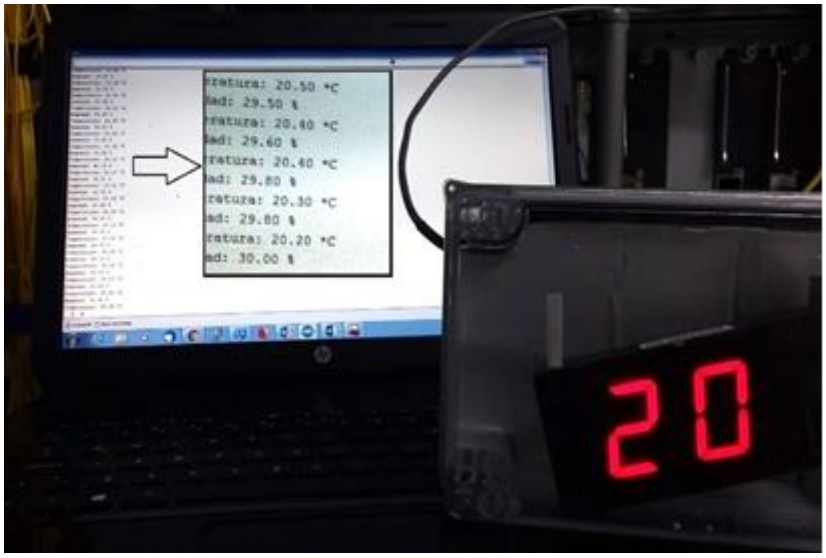


Figura 3.32: Toma de valores DHT

Tabla 3.2: Valores de temperatura y humedad medidos por el DHT

TEMPERATURA DHT (^{O}C)	HUMEDAD % HR	TEMPERATURA TERMÓMETRO (^{O}C)	OBSERVACIÓN
20,60	29,60	20	Temperaturas similares
20,50	29,50	20	Temperaturas similares
20,40	29,60	20	Temperaturas similares
20,30	29,80	20	Temperaturas similares
20,20	30	20	Temperaturas similares
20,10	30,20	20	Temperaturas similares
20,20	30	20	Temperaturas similares

- **Sensor MQ2**

El sensor de gas MQ2 tiene un precalentamiento de 24 horas y fue necesario esperar este tiempo para obtener temperaturas estables. Este sensor es usado para detectar principalmente gases como, el humo causado por incendio, flamas y chispas eléctricas. En la figura 3.33, se puede observar los valores medidos en el monitor serial.

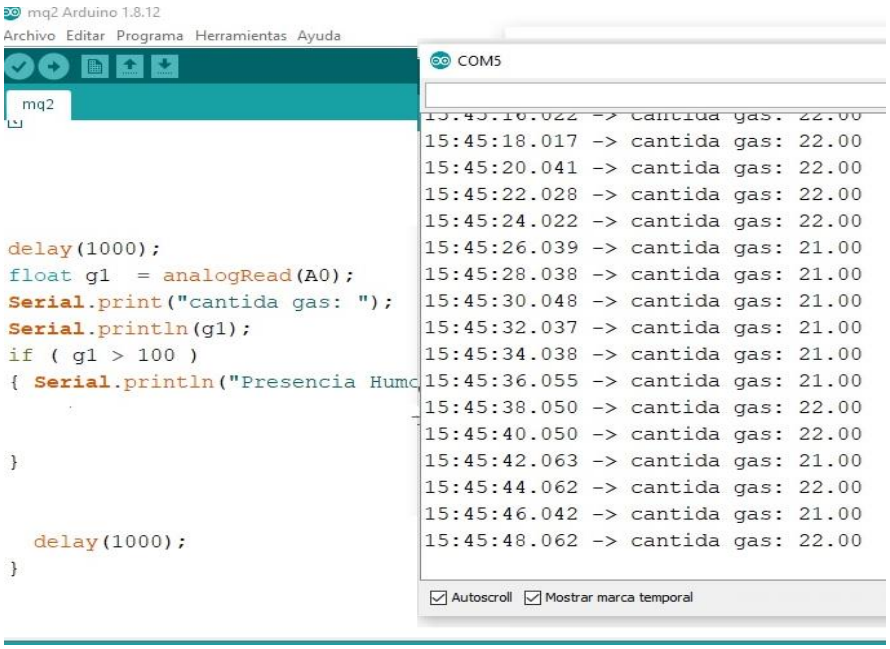


Figura 3.33: Toma de valores MQ2

En la figura 3.34, se puede observar la creación del mensaje de alerta cuando es detectado humo. En la tabla 3.3, se muestra algunos valores medidos por un sensor MQ2 en el monitor serial.

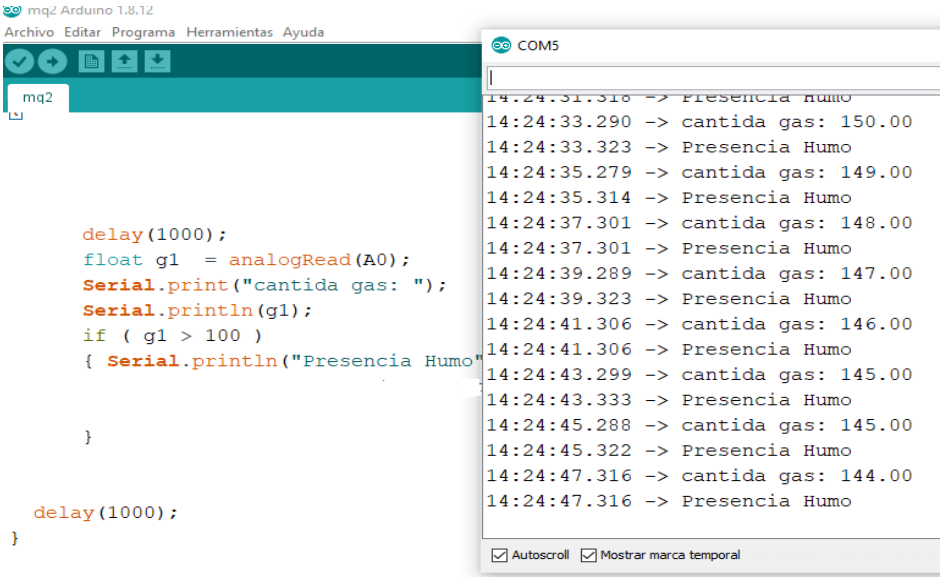


Figura 3.34: Mensaje de alerta ante la presencia de humo

Tabla 3.3: Valores de concentración de gas en el ambiente

CONCENTRACIÓN DE GAS (PPM)	MENSAJE
21	N/A
22	N/A
144	Presencia de humo
145	Presencia de humo

- **Sensores de Corriente**

Se utilizó sensores no invasivos para no poner en riesgo la infraestructura eléctrica de Telconet. La señal de los sensores es enviada al circuito de acondicionamiento. Los valores a la salida del circuito de acondicionamiento son enviados a los pines analógicos del Arduino. Con estos datos se procedió a realizar pruebas de medición del sensor (figura 3.35). En la tabla 3.4, se muestra algunos valores registrados por un sensor SCT-013 y los valores medidos con una pinza amperimétrica.

Figura 3.35: Toma de valores SCT-013

Tabla 3.4: Valores de corriente medidos por el SCT-013

CORRIENTE SCT (A)	CORRIENTE PINZA (A)	DESCRIPCIÓN	OBSERVACIÓN
8.9	8,12	B1 tablero distribución cuarto UPS	Valores similares
8.11	8,12	B1 tablero distribución cuarto UPS	Valores similares
12.5	12.8	B3 tablero distribución cuarto UPS	Valores similares
12.7	12.8	B3 tablero distribución cuarto UPS	Valores similares
1	1.3	B7 tablero distribución cuarto UPS	Valores similares

CORRIENTE SCT (A)	CORRIENTE PINZA (A)	DESCRIPCIÓN	OBSERVACIÓN
1.1	1.3	B7 tablero distribución cuarto UPS	Valores similares

- **Sensores de flama**

La prueba se realizó con la ayuda de una flama de un fósforo con el objetivo de comprobar si es detectada por el sensor. En la figura 3.36, se puede observar la toma de valores registrados en el monitor serial. En la tabla 3.5, se muestra los valores registrados por un sensor de flama.

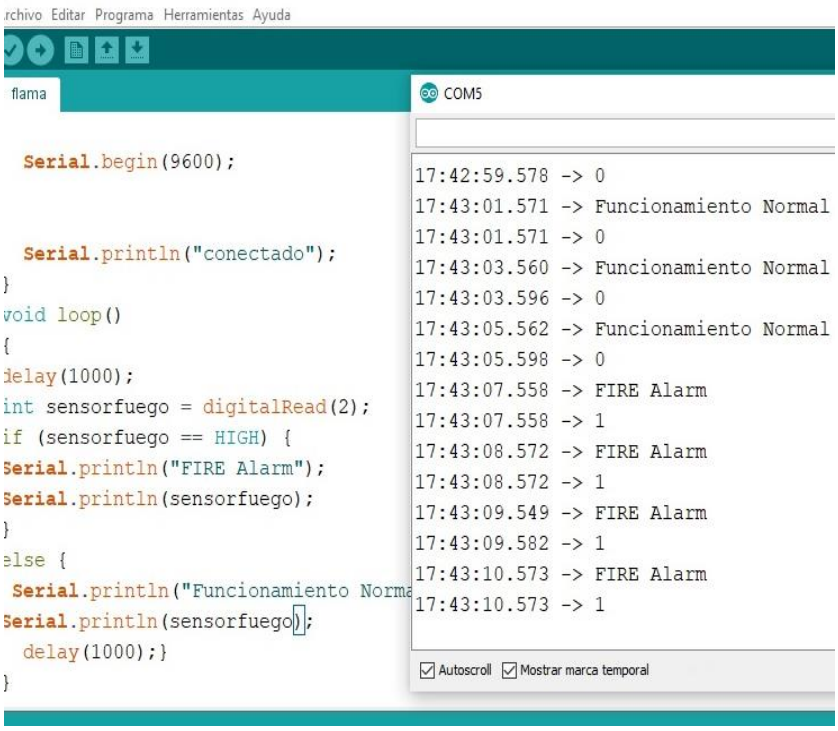


Figura 3.36: Toma de valores sensor de flama

Tabla 3.5: Valores medidos por el sensor de flama

VALOR	MENSAJE	DESCRIPCIÓN
0L	Funcionamiento Normal	S/N
0L	Funcionamiento Normal	S/N
1L	*FIRE Alarm*	Presencia de fuego
1L	*FIRE Alarm*	Presencia de fuego

- **Sensor magnético**

La prueba se realizó abriendo y cerrando la puerta del nodo. En la figura 3.37, se puede observar la toma de valores registrados en el monitor serial. En la tabla 3.6, se muestra los valores registrados por un sensor magnético.

Figura 3.37: Toma de valores sensor magnético

Tabla 3.6: Valores medidos por el sensor magnético

VALOR	MENSAJE	DESCRIPCIÓN
0L	Puerta cerrada	Puerta cerrada
0L	Puerta cerrada	Puerta cerrada
1L	Puerta Abierta	Puerta Abierta
1L	Puerta Abierta	Puerta Abierta

Prueba de funcionamiento de la API

La prueba se realizó usando el programa POSTMAN, este programa permite realizar peticiones HTTP a la API. El primer paso antes del realizar las pruebas, fue iniciar la API.

```
python api.py
```

Para iniciar la API, se escribió la línea de comando en una consola Python (figura 3.38). *Flask* devolvió una serie de mensajes para indicar que el servidor inició sin ningún problema.

```
(venv) C:\Users\dafra\OneDrive\Escritorio\apirest2>python api.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 912-499-997
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

Figura 3.38: Inicio del servidor

La primera prueba que se realizó fue el envío de valores de temperatura a la base de datos (figura 3.39). En la sección *body* del programa se escribió los datos que la API espera recibir, además se especificó la URL y la petición HTTP.

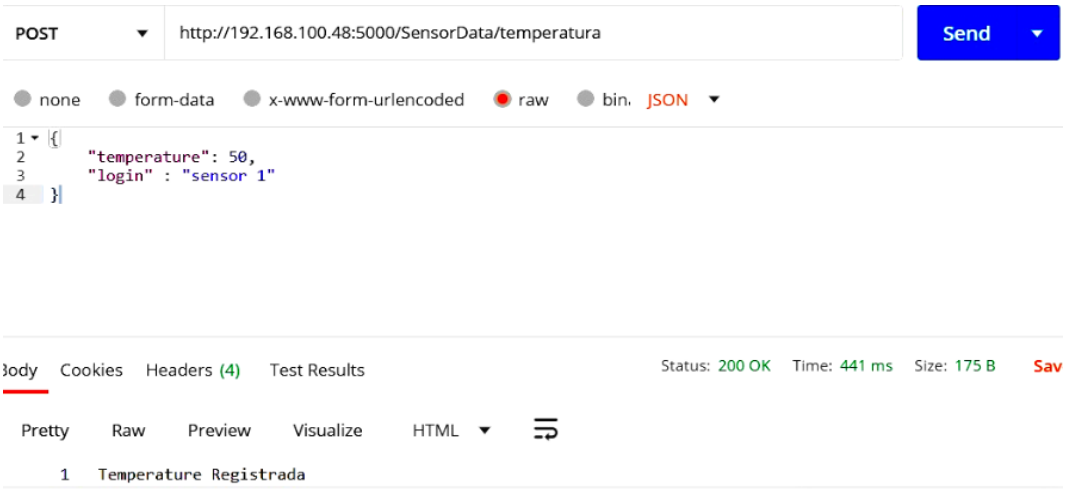


Figura 3.39: Valores de temperatura registrados en la base de datos

El programa recibió una respuesta 200 (respuesta correcta) y el mensaje “Temperatura Registrada” (mensaje configurado en el fichero api.py), como confirmación que la solicitud HTTP fue exitosa.

La siguiente prueba que se realizó fue el envío de mensajes de alerta a la base de datos. Para realizar esa petición en el *body* del programa únicamente se escribió un solo dato (*log*). Este dato es el único que la API espera recibir en esta URL. Como se observa en la figura 3.40, se recibió una respuesta 200 y el mensaje “Evento Registrado”.

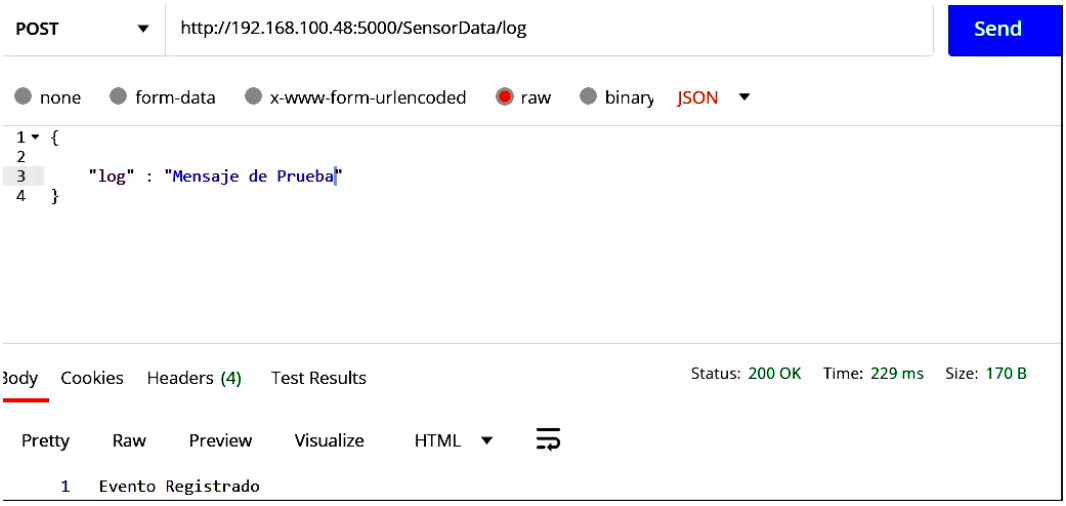


Figura 3.40: Mensajes de alerta registrados en la base de datos

Se hizo la misma prueba a todas las rutas que utilizan el método POST, en cada prueba se recibió un código 200 y los mensajes de respuesta correspondientes.

La siguiente prueba que se realizó fue obtener información desde la base de datos. Lo único que se especificó para realizar una petición GET fue la URL de la API. Se recibió una respuesta 200. Como se observa en la figura 3.41, se obtuvo los valores de humedad almacenados en la base de datos.

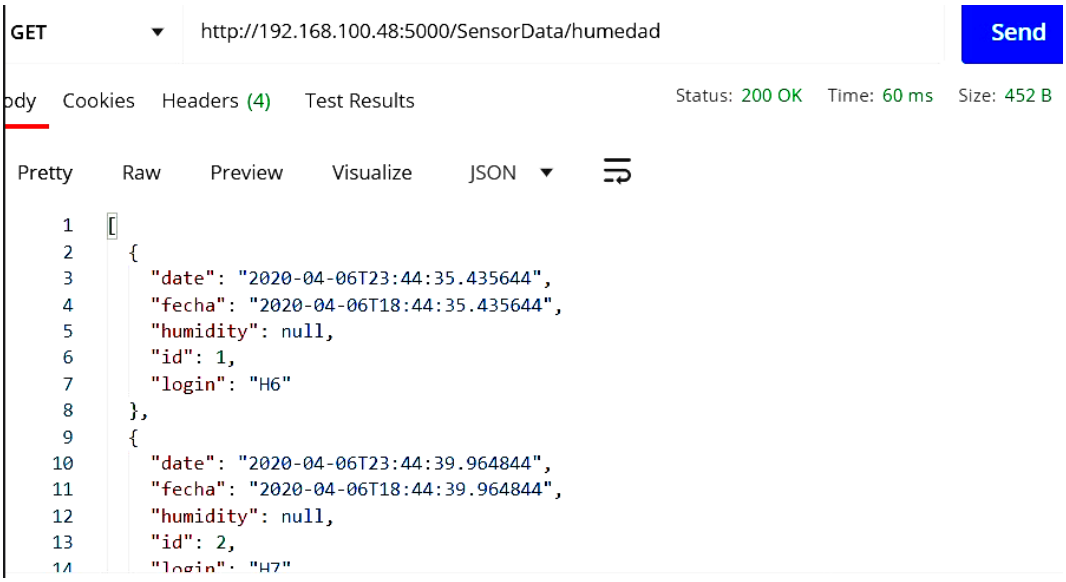


Figura 3.41: Valores de humedad desde la base de datos

La siguiente prueba que se realizó fue obtener valores de voltaje almacenados en la base de datos. Como se observa en la figura 3.42, se recibió una respuesta 200 y se obtuvo los valores de voltaje.

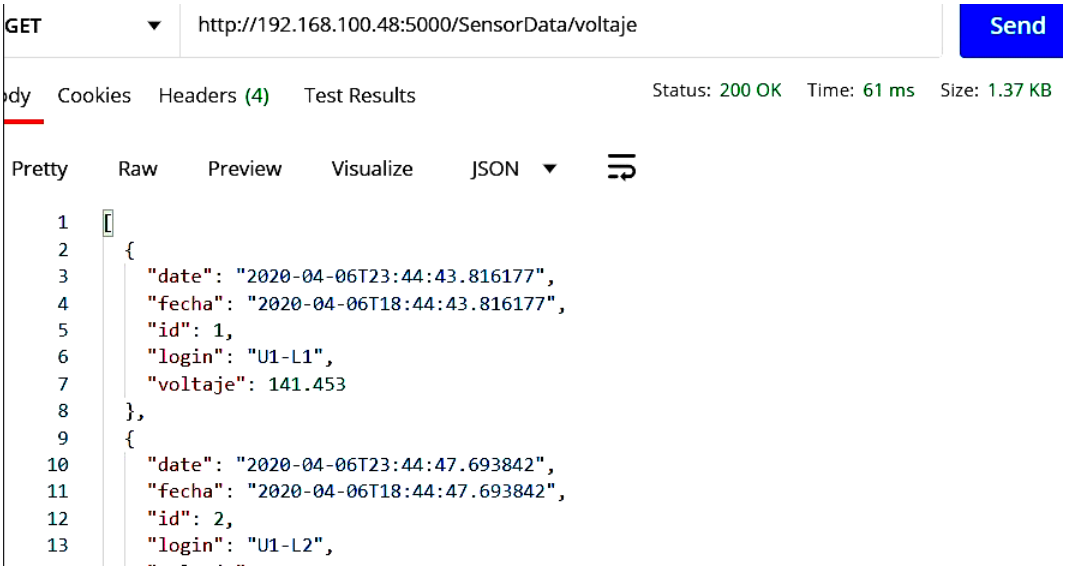


Figura 3.42: Valores de voltaje desde la base de datos

Se hizo la misma prueba a todas las rutas que utilizan el método GET, en cada prueba se recibió un código 200 y los valores de los sensores correspondientes.

Prueba de comunicación entre Arduino y la base de datos.

Los Arduinos envían *información* a la base de datos, utilizando como intermediario la API. La prueba se la realizó conectando todos los elementos a la red. Cada uno de los Arduinos envió una petición POST a la API. La API aceptó la solicitud y verificó que los datos estén autenticados. La API retornó a los Arduinos una respuesta HTTP 200 por cada petición ejecutada correctamente (figura 3.43).

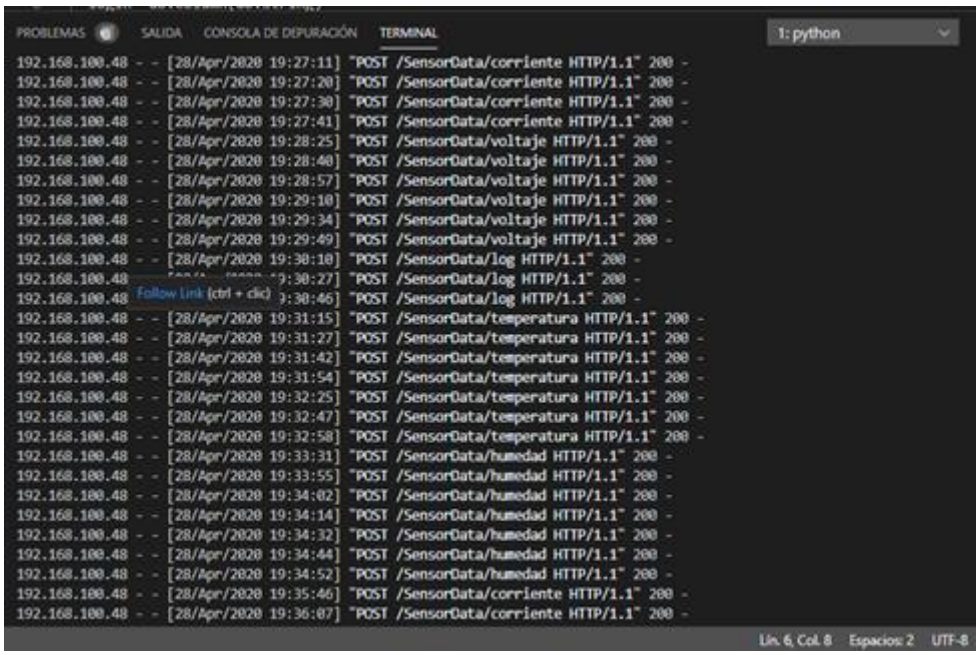


Figura 3.43: Envío de información a la API

Para corroborar que la API realizó el proceso correctamente, se verificó que la información esté almacenada en la base de datos. Hay dos opciones para realizar esta verificación. La primera fue acceder a una URL de cualquier ruta que se definió en el fichero api.py. El navegador hizo una petición GET a la API, la API aceptó la solicitud y solicitó autenticación para extraer la información de la base de datos y enviarla al navegador. Se verificó el funcionamiento de todas las rutas y se obtuvo resultados satisfactorios (figura 3.44). La ruta que se eligió para constancia del funcionamiento, permite visualizar los valores de temperatura en formato JSON, como se muestra en la figura 3.45.

```
192.168.100.48 - - [28/Apr/2020 20:44:52] "GET /SensorData/temperatura HTTP/1.1" 200 -
192.168.100.48 - - [28/Apr/2020 20:44:59] "GET /SensorData/humedad HTTP/1.1" 200 -
192.168.100.48 - - [28/Apr/2020 20:45:04] "GET /SensorData/gas HTTP/1.1" 200 -
192.168.100.48 - - [28/Apr/2020 20:45:09] "GET /SensorData/corriente HTTP/1.1" 200 -
192.168.100.48 - - [28/Apr/2020 20:45:17] "GET /SensorData/voltaje HTTP/1.1" 200 -
192.168.100.48 - - [28/Apr/2020 20:45:22] "GET /SensorData/log HTTP/1.1" 200 -
```

Figura 3.44: Peticiones GET de todas las URLs

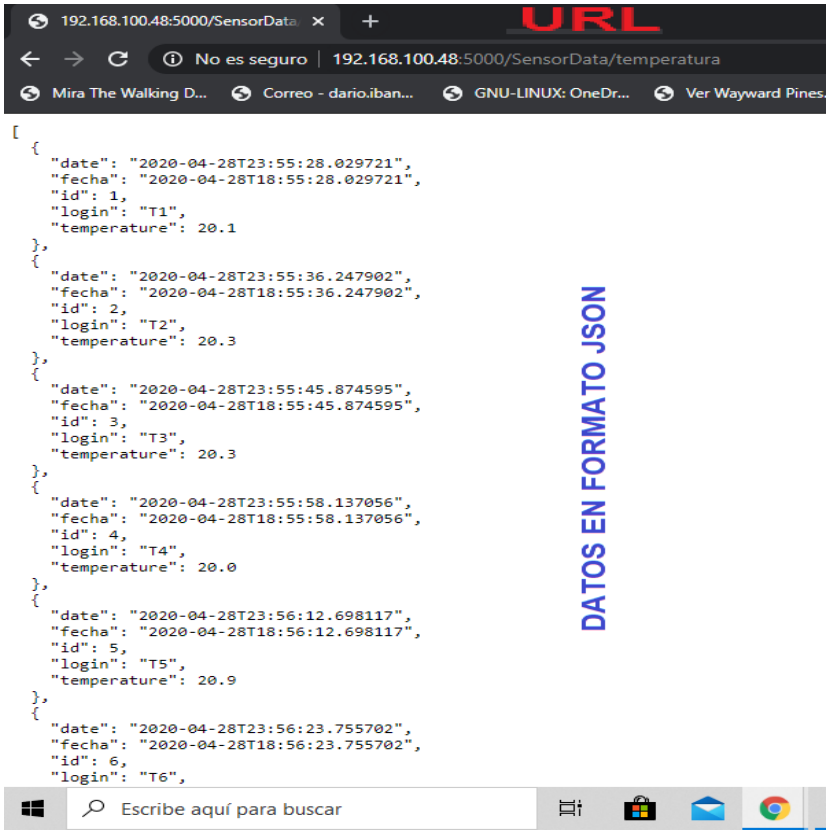
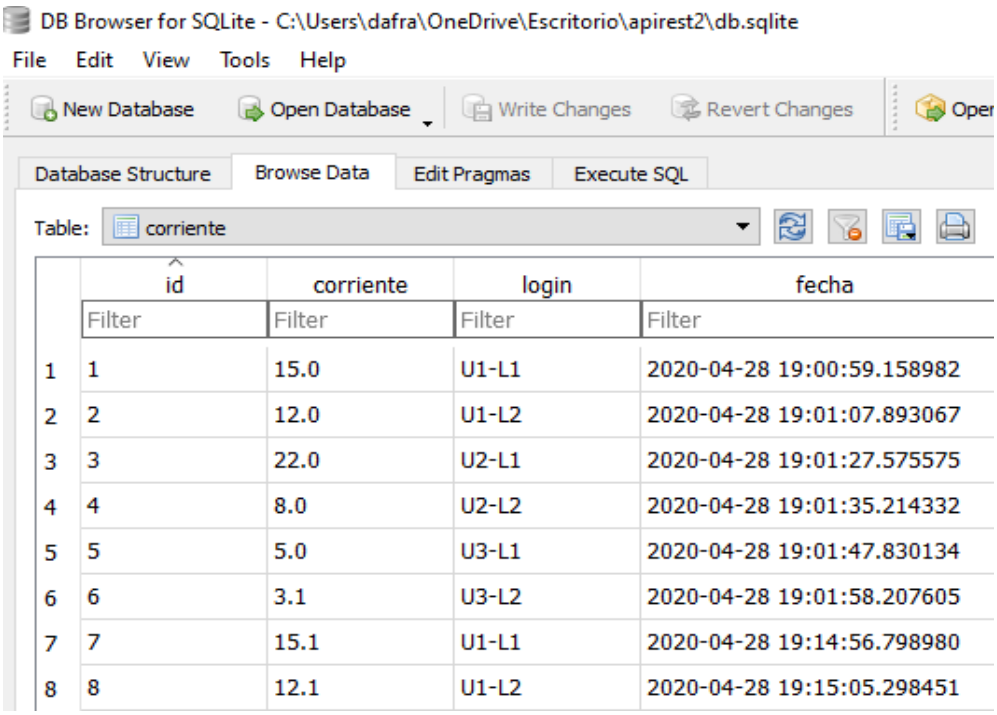


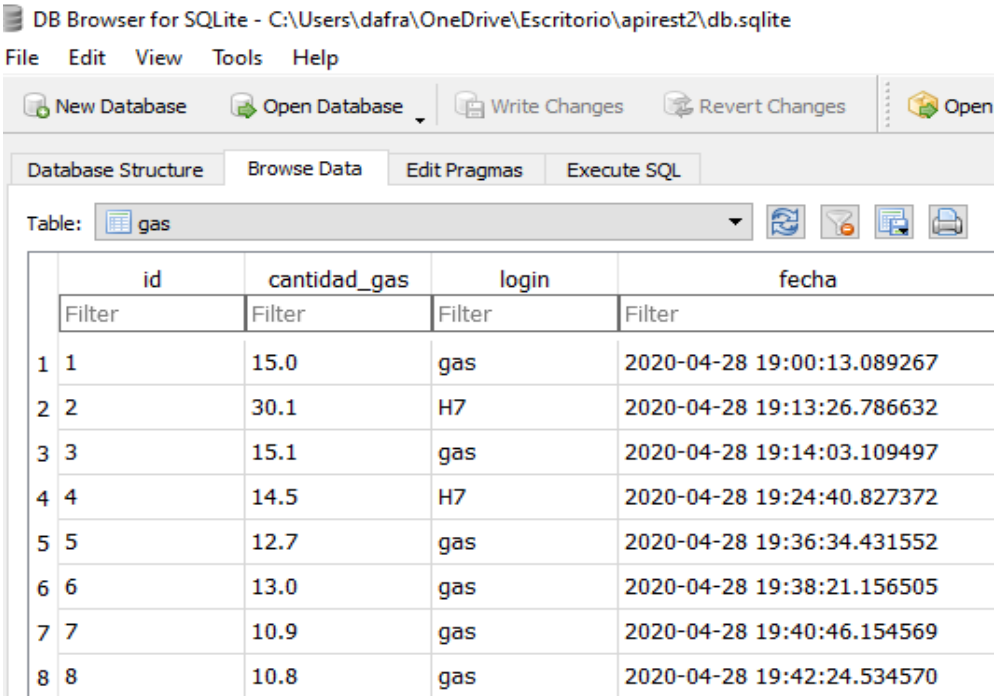
Figura 3.45: URL: http://192.168.100.48:5000/SensorData/temperatura

La segunda opción fue verificar directamente en la base de datos la información almacenada. Para esta opción, se ingresó a la base de datos mediante *SQLITE BROWSER* y se verificó que los datos estén almacenados. En las figuras 3.46, 3.47, 4.48, 3.49, 3.50 y 3.51 se presenta las tablas de la base de datos.



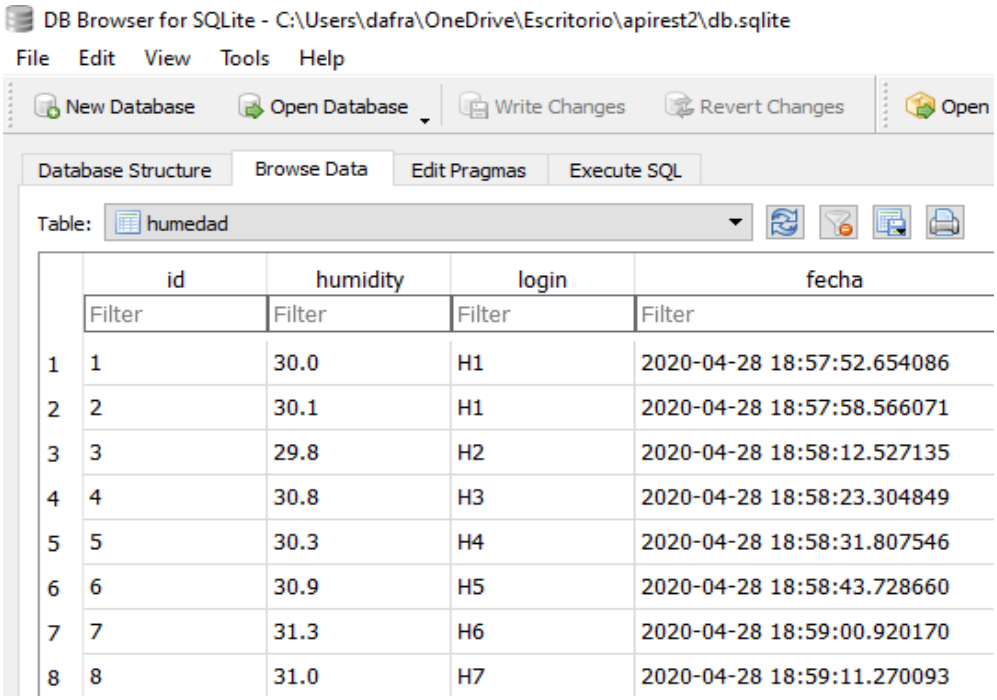
	id	corriente	login	fecha
	Filter	Filter	Filter	Filter
1	1	15.0	U1-L1	2020-04-28 19:00:59.158982
2	2	12.0	U1-L2	2020-04-28 19:01:07.893067
3	3	22.0	U2-L1	2020-04-28 19:01:27.575575
4	4	8.0	U2-L2	2020-04-28 19:01:35.214332
5	5	5.0	U3-L1	2020-04-28 19:01:47.830134
6	6	3.1	U3-L2	2020-04-28 19:01:58.207605
7	7	15.1	U1-L1	2020-04-28 19:14:56.798980
8	8	12.1	U1-L2	2020-04-28 19:15:05.298451

Figura 3.46: Tabla de corriente



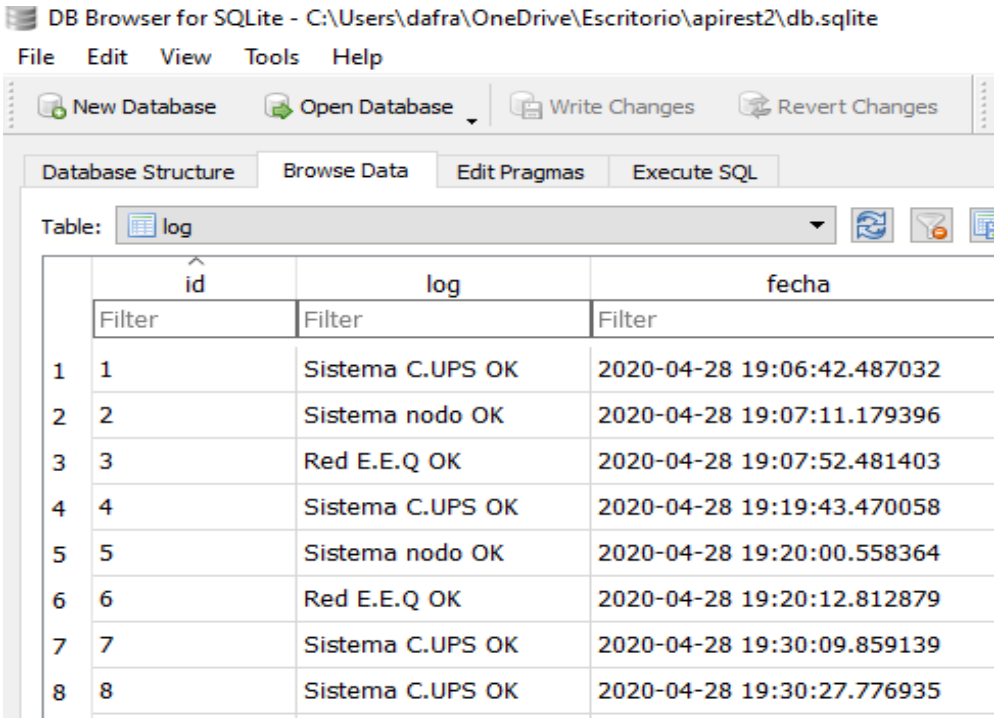
	id	cantidad_gas	login	fecha
	Filter	Filter	Filter	Filter
1	1	15.0	gas	2020-04-28 19:00:13.089267
2	2	30.1	H7	2020-04-28 19:13:26.786632
3	3	15.1	gas	2020-04-28 19:14:03.109497
4	4	14.5	H7	2020-04-28 19:24:40.827372
5	5	12.7	gas	2020-04-28 19:36:34.431552
6	6	13.0	gas	2020-04-28 19:38:21.156505
7	7	10.9	gas	2020-04-28 19:40:46.154569
8	8	10.8	gas	2020-04-28 19:42:24.534570

Figura 3.47: Tabla de gas



	id	humidity	login	fecha
	Filter	Filter	Filter	Filter
1	1	30.0	H1	2020-04-28 18:57:52.654086
2	2	30.1	H1	2020-04-28 18:57:58.566071
3	3	29.8	H2	2020-04-28 18:58:12.527135
4	4	30.8	H3	2020-04-28 18:58:23.304849
5	5	30.3	H4	2020-04-28 18:58:31.807546
6	6	30.9	H5	2020-04-28 18:58:43.728660
7	7	31.3	H6	2020-04-28 18:59:00.920170
8	8	31.0	H7	2020-04-28 18:59:11.270093

Figura 3.48: Tabla de humedad



	id	log	fecha
	Filter	Filter	Filter
1	1	Sistema C.UPS OK	2020-04-28 19:06:42.487032
2	2	Sistema nodo OK	2020-04-28 19:07:11.179396
3	3	Red E.E.Q OK	2020-04-28 19:07:52.481403
4	4	Sistema C.UPS OK	2020-04-28 19:19:43.470058
5	5	Sistema nodo OK	2020-04-28 19:20:00.558364
6	6	Red E.E.Q OK	2020-04-28 19:20:12.812879
7	7	Sistema C.UPS OK	2020-04-28 19:30:09.859139
8	8	Sistema C.UPS OK	2020-04-28 19:30:27.776935

Figura 3.49: Tabla de mensajes

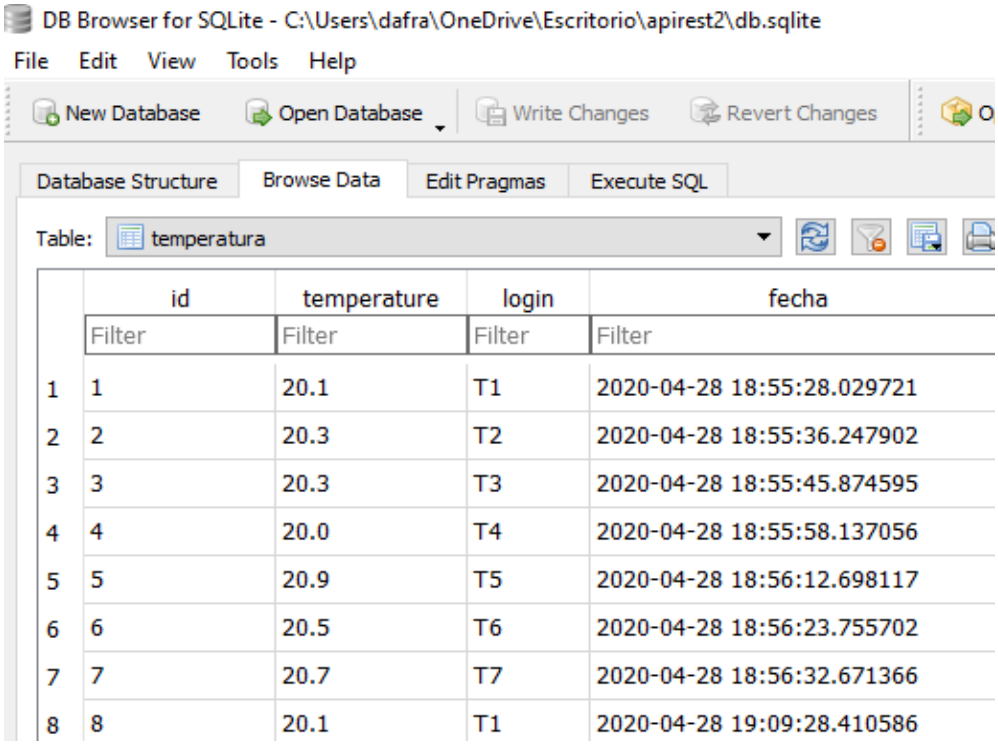


Figura 3.50: Tabla de temperaturas

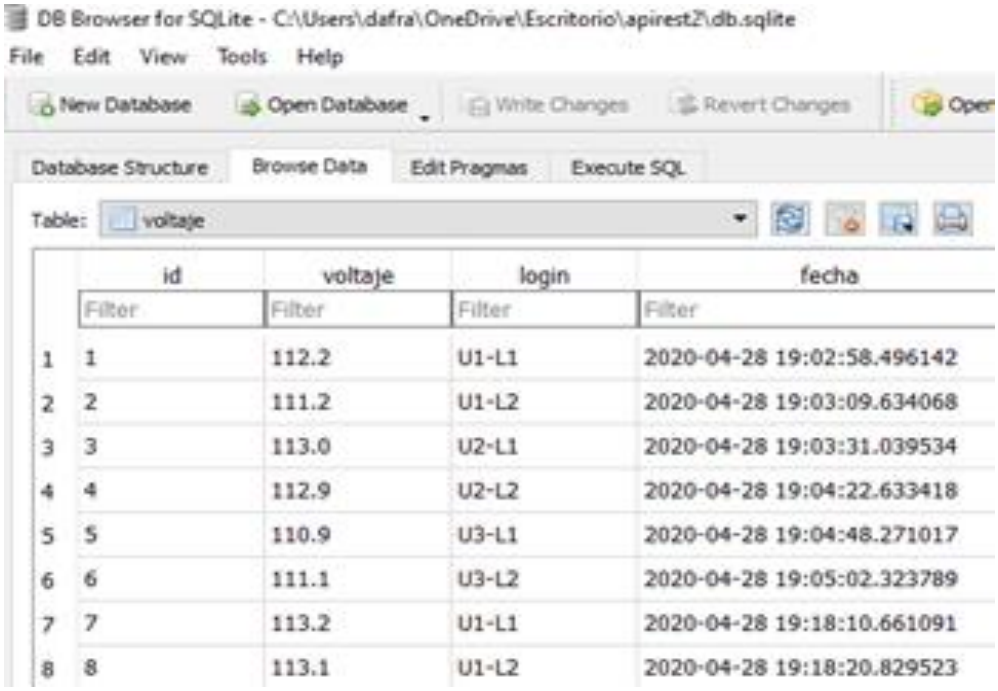


Figura 3.51: Tabla de voltajes

Prueba de funcionamiento de la página *web*

La prueba se realizó accediendo a la página *web* utilizando cualquier navegador. Se digitó en la barra de navegación la URL 192.168.XXX.XXX. La primera página que mostró el sitio *web* fue la de autentificación (figura 3.52).

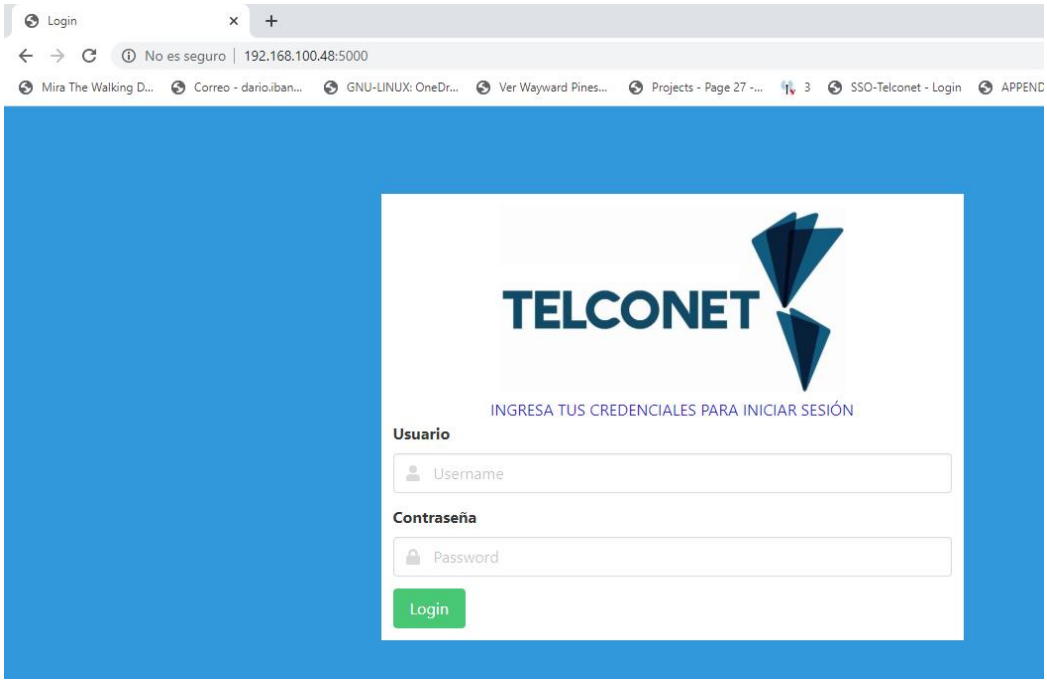


Figura 3.52: Página de autentificación

Se ingresó la credenciales, posterior el sitio *web* redirigió a la página *home* (figura 3.53).

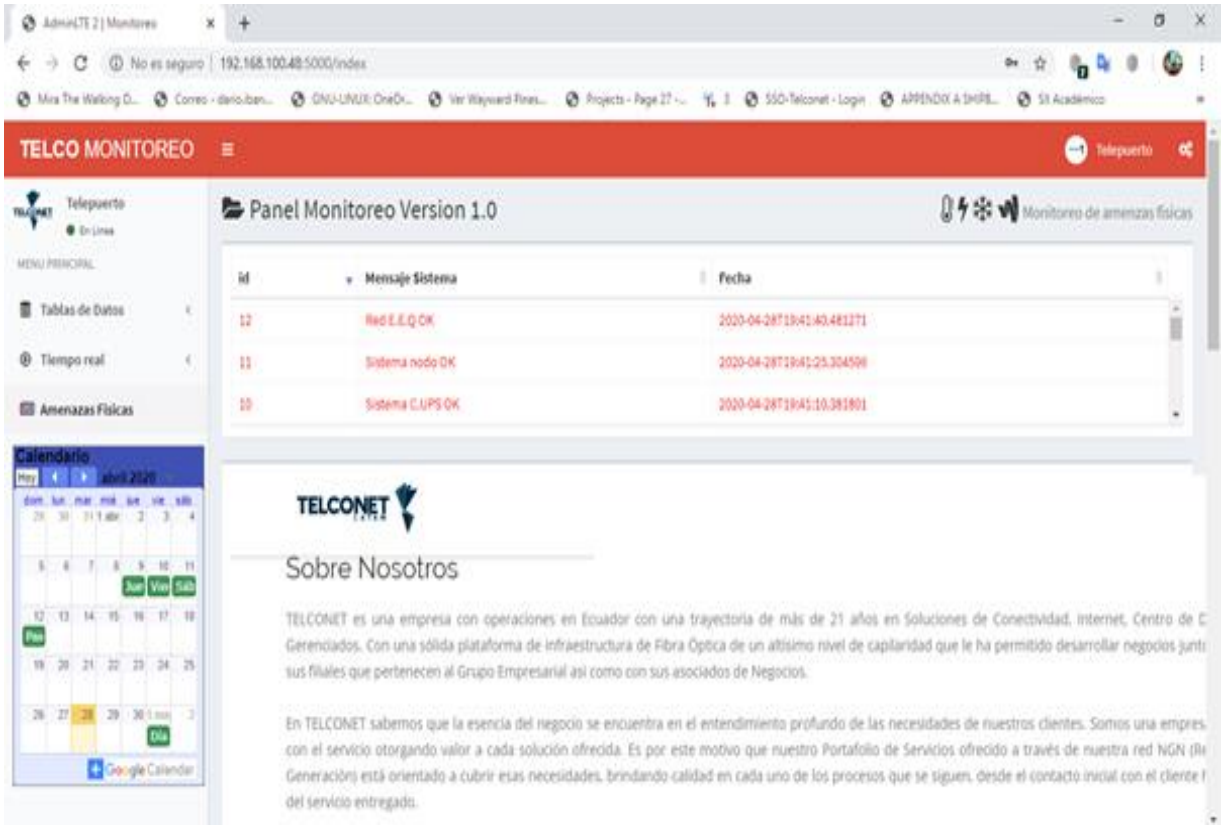


Figura 3.53: Página de bienvenida

Se comprobó el funcionamiento de todas las páginas del sitio *web.* Cada página mostró la información correspondiente. La página que se eligió para constancia del funcionamiento,

permite visualizar los valores de corriente recolectados por todos los sensores SCT en una tabla (figura 3.54).

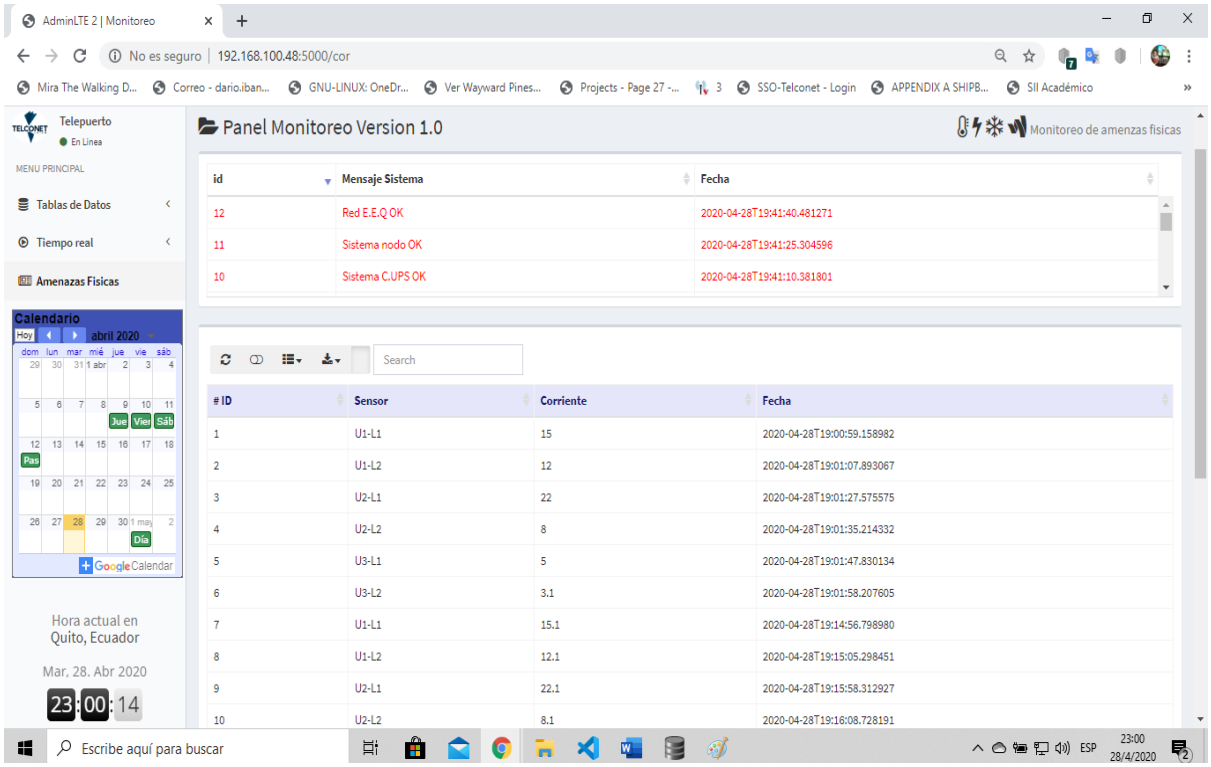


Figura 3.54: Página de corrientes

Prueba de funcionamiento del sistema integrado

Para realizar la prueba final se cambió el servidor de la fase de desarrollo a la fase de producción. Esto se realizó eliminando el parámetro *debug* en el fichero api.py.

```
app.run(host="0.0.0.0")
```

La prueba se realizó verificando el correcto funcionamiento del sistema por 24 horas. En esta prueba se comprobó que los valores recolectados por todos los sensores fueran enviados correctamente por los Arduinos y almacenados en la base de datos. Se navegó por las distintas páginas del sitio *web,* las cuales en todo momento mostraron la información que se deseaba visualizar. Se verificó que los mensajes del sistema se actualicen automáticamente en la tabla creada para mostrar la información del sistema. Adicionalmente, se comprobó que los mensajes de alerta sean generados y enviados correctamente y que estos se visualicen en tiempo real en la tabla creada para mostrar la información del sistema. Por último, se verificó que las páginas del sitio *web* que muestran gráficos en tiempo real (figura 3.55 y 5.56), muestren esta información correctamente.

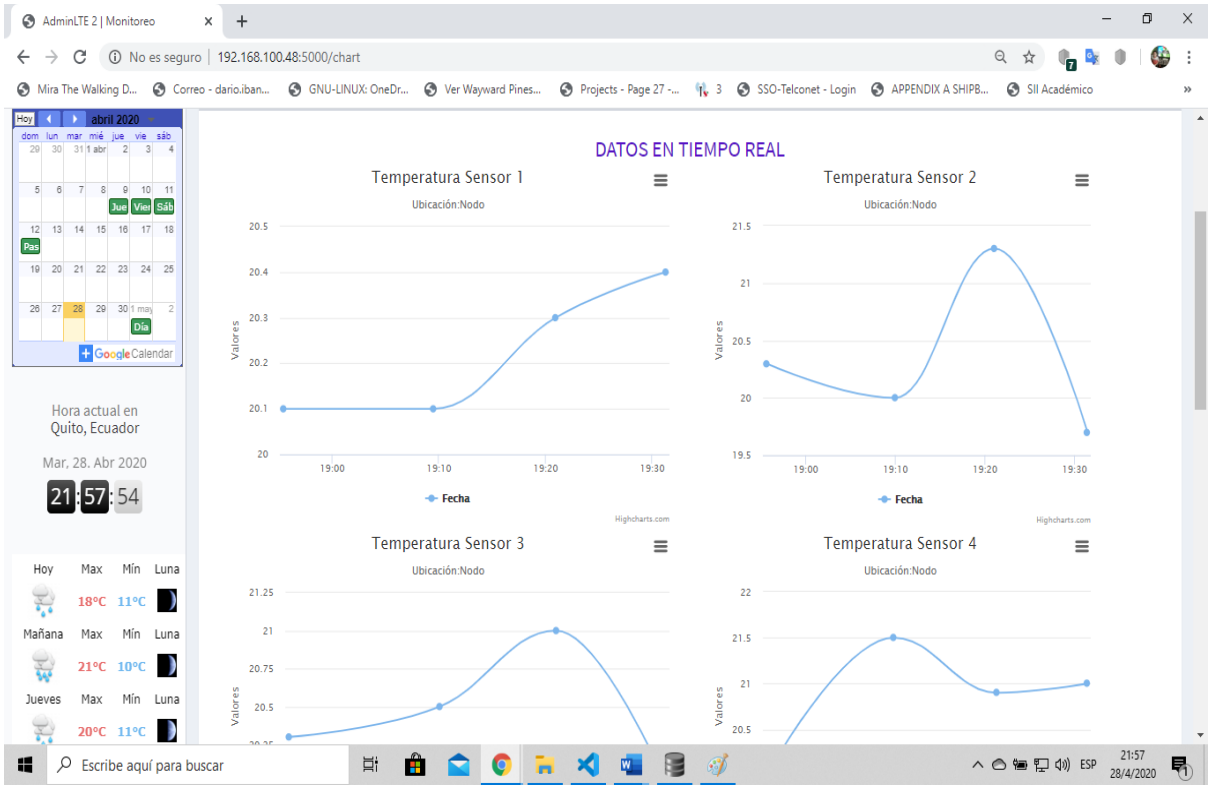


Figura 3.55: Página de lectura de temperatura en tiempo real

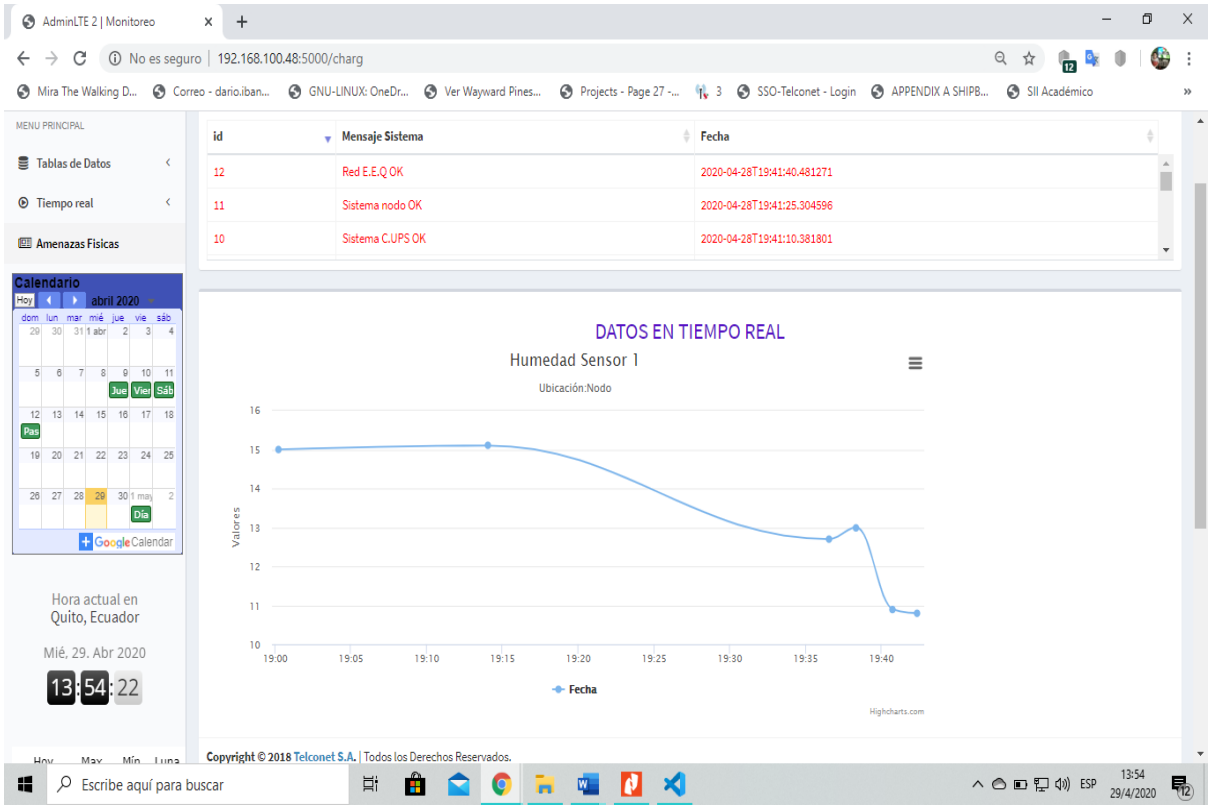


Figura 3.56: Página de lectura de gas en tiempo real

4. CONCLUSIONES Y RECOMENDACIONES

4.1. Conclusiones

- Se implementó un sistema de monitoreo de amenazas físicas en el cuarto de telecomunicaciones del nodo Gosseal de Telconet, en un lapso de 4 meses. Con este fin, se desplegó una red LAN exclusiva para el envío constante de valores a una base de datos. Los elementos que componen el sistema son: 26 sensores de tipo flama, temperatura, humedad, 3 Arduinos, una Raspberry Pi 3b+ y un *switch* encargado de la interconexión de los dispositivos. Los Arduinos adquieren, procesan y envían la información a la base de datos. Por su parte, la Raspberry Pi aloja la base de datos y la API REST. La API es el intermediario que permite la comunicación entre los Arduinos y la base de datos.

- Se realizó un estudio de las amenazas físicas, mediante una visita técnica a la mini central de telecomunicaciones, donde se estableció que las amenazas físicas que la afectan son: amenazas a los equipos de telecomunicaciones, fugas de aire frío, humo o incendios y fallos en los suministros eléctricos.

- Se diseñó e implementó el sistema de monitoreo, donde por medio de un plano se especificó la ubicación de: sensores, Arduinos, Raspberry Pi y canalización. Los sensores fueron ubicados de acuerdo con los parámetros que éstos pueden medir y los Arduinos fueron ubicados de manera central con respecto a los sensores.

- Se desarrolló una API que actúa como *software* de control, ésta es la encargada de gestionar las peticiones POST que solicitan los Arduinos, las peticiones GET que solicita el cliente y realizar consultas a la base de datos.

- Se evaluó el correcto funcionamiento del sistema, mediante pruebas que permitieron verificar la correcta comunicación entre los Arduinos, la base de datos y el sitio *web*. Los Arduinos enviaron correctamente los valores de los sensores a la base de datos y estos valores pudieron ser observados en las páginas del sitio *web*, sin ningún inconveniente.

- El Internet de las cosas (IoT), es una tecnología que permite en la actualidad la conexión a internet de casi todos los dispositivos. Con esta tendencia, se puede esperar que en unos cuantos años se fabriquen dispositivos con la capacidad de conectarse directamente a internet y se obtengan entornos totalmente automatizados.

- Con el diseñó de un sistema de monitoreo en una mini central, se ha iniciado una base para que Telconet comience a implementar este sistema en las demás mini centrales, con el fin de evitar tiempos inactivos en la producción.

- Al desarrollar cada una de las pruebas de funcionamiento (prueba de funcionamiento de sensores, prueba de funcionamiento de la API prueba de comunicación entre Arduino y la base de datos, prueba de consulta de información mediante la página web), se determinó que la comunicación entre todos los componentes de la red de monitoreo se realiza de forma correcta, ya que no presentó errores de comunicación.

- Las placas Arduino son las más utilizadas para realizar proyectos de control, gracias a su sencillez y bajo costo, pero sobre todo a la gran cantidad de sensores y dispositivos compatibles que hay en el mercado. Además, permiten crear proyectos novedosos ya que los desarrolladores constantemente añaden mejoras creando nuevas *shield*, placas o librerías.

- Cuando se alimenta a los sensores con una fuente externa, se debe interconectar las conexiones a tierra para que el Arduino pueda tomar las lecturas correctas.

- Python es uno de los lenguajes más utilizados para desarrollar proyectos en el mundo, principalmente porque la curva de aprendizaje es baja. Además, permite escribir el código de forma simple y elegante. Y cuenta con una gran cantidad de librerías y módulos que permiten realizar cualquier tipo de proyecto. En el proyecto se utilizaron librerías como *Flask* y sus extensiones *Flask RESTful, Flask SQLAlchemy*, *Flask Marshmallow* que sirvieron para configurar la *APIREST* y crear la base de datos.

- La tarjeta Raspberry Pi 3B+, al ser un mini ordenador de gran potencia, fue usada como servidor para el sistema de monitoreo. Permanece encendida y conectada a la red, lo que permite enviar y consultar información en la red local en cualquier instante que se necesite.

- Los sensores digitales permiten obtener datos más precisos y alcanzar distancias desde pocos centímetros hasta 20 metros para la comunicación con el microcontrolador, los sensores utilizados en el proyecto (DHT22, MQ2, magnético) son elaborados con el fin de ser usados por las placas Arduino, por tal razón no necesitan de un circuito de acondicionamiento. Debido a esta característica y al valor en el mercado, hacen que sean los más usados en proyectos de electrónica.

- El sensor digital de temperatura y humedad (DHT22) utiliza un solo pin para la transmisión de ambas magnitudes. Para esta comunicación se utiliza un total de 80 bits de los cuales, los primeros 16 corresponden a los valores de humedad relativa, los siguientes 16 a valores de humedad y los 8 bits restantes para verificación.

- Los sensores de corriente no invasivos son utilizados en proyectos en los cuales no se puede intervenir la infraestructura eléctrica (cortar o desempalmar). Para obtener una lectura correcta se debe cerrar el núcleo ferromagnético adecuadamente y por el solo se debe atravesar un solo conductor.

4.2. Recomendaciones

- Es necesario realizar la verificación de la versión de los paquetes utilizados en la creación de un código de un proyecto debido a que, en la mayoría de los casos cada versión usa una sintaxis diferente.

- Se recomienda no modificar o alterar la información alojada en la base de datos cuando esté recibiendo o enviando información a través de la API; debido a que, la base de datos puede corromperse o puede ser considerada como sospechosa.

- Es importante considerar que, al iniciar el sistema, primero deben estar encendidos los Arduinos y después proceder con el encendido de los sensores ya que, si sucede lo contrario el Arduino no toma datos.

- Para energizar los Arduinos es necesario usar fuentes de alimentación que proporcionen voltajes entre 5 a 9 voltios. Energizar los Arduinos con voltajes superiores ocasiona sobrecalentamiento en las placas.

- En lugares en donde se necesite un mayor número de sensores, se recomienda usar una placa con mayor número de pines como la placa Arduino Mega, con el fin de evitar que algún sensor se quede sin conexión.

- Asignar una IP fija a la Raspberry Pi evitará que, en caso de producirse un reinicio en los componentes del sistema, no se tenga que volver a configurar los programas de los mismos.

5. REFERENCIAS BIBLIOGRAFÍCAS

[1] . J. Maldonado Mahauad, “DISEÑO DE UN CENTRO DE DATOS BASADO EN ESTANDARES. CASO PRÁCTICO: DISEÑO DEL CENTRO DE DATOS DEL COLEGIO LATINOAMERICANO”, Cuenca : Universidad de Cuenca - Facultad de Ingeniería - Escuela de Informática, 2010.

[2] C. Cowan y C. Gaskins , «Monitoreo de amenazas físicas en centros de datos,» *Schneider Electric – Data Center Science Center,* vol. Documento Técnico 102, nº Rev3, pp. 2 - 8, 2012.

[3] Omega, «OMEGA,» [En línea]. Available: https://cl.omega.com/prodinfo/instrumentacion.html.. [Último acceso: 21 09 2019].

[4] M. Gutiérrez y S. Iturralde, «INTRODUCCIÓN A LA INSTRUMENTACIÓN Y NORMAS,» de *FUNDAMENTOS BASICOS DE INSTRUMENTACION Y CONTROL*, Santa Elena, Universidad Estatal Península de Santa Elena, 2017, pp. 1, 2.

[5] A. Everlet y J. Pastor , «INTRODUCCIÓN AL INTERNET DE LAS COSAS,» *Carriots,* pp. 13 -27, 2013.

[6] H. Gómez Rodríguez, U. Dávalos Guzmán , R. Martínez Atilano , N. Bautista Rangel y M. Jiménez Rodríguez , «Computadoras de placa reducida Raspberry Pi 3 y Asus Tinker Board,» *Revista Iberoamericana de las Ciencias Computacionales,* vol. 7, nº 14, 2018.

[7] Raspberrypi, «Raspberrypi.org,» [En línea]. Available: https://www.raspberrypi.org/. [Último acceso: 6 10 2019].

[8] Raspberry Pi Foundation, «Raspberry Pi 3 Model B+,» *Raspberry Pi,* pp. 2, 3.

[9] Cytron Technologies, «Cytron Technologies,» 25 07 2018. [En línea]. Available: https://tutorial.cytron.io/2018/07/25/raspberry-pi-3b-vs-3b/. [Último acceso: 2 10 2019].

[10] C. González Domínguez y B. Palomo Vázquez, Aplicaciones orientadas a la domótica con Raspberry Pi, Sevilla: Universidad de Sevilla - Escuela Técnica Superior de Ingeniería - Departamento de Ingeniería Electrónica, 2015.

[11] E. Upton y G. Halfacree, «GUÍA DEL USUARIO,» *R A S P B E R R Y P I.*

[12] D. Espla, «eloutput.com,» 23 01 2020. [En línea]. Available: https://eloutput.com/input/guia-compras/mejor-tarjeta-microsd-comprar/. [Último acceso: 09 05 2020].

[13] Arduino, «Arduino,» 2019. [En línea]. Available: https://www.arduino.cc/en/Guide/Introduction#. [Último acceso: 5 10 2019].

[14] J. C. Herrero Herranz y J. Sanchéz Allende, «UNA MIRADA AL MUNDO ARDUINO,» *TECNOLOGÍ@ Y DESAROLLO,* vol. XIII, pp. 5 - 8, 2015.

[15] J. Garrido Pedraza, Fundamentos de Arduino, 2015.

[16] Ó. Torrente Artero, «HARDWARE ARDUINO,» de *ARDUINO Curso práctico de formación*, Madrid, Alfaomega-RCLibros, 2013, pp. 116,117.118.

[17] J. Mayné , «Sensores Acondicionadores y Procesadores de Señal,» *SILICA,* nº Rev.2, p. 4, 2003.

[18] A. Serna Ruiz, A. F. Ros García y J. C. Rico Noguera, «Los Sensores y su Clasificación,» de *Guía práctica de sensores*, Creaciones Copyright S.L, 2010, pp. 5 -7.

[19] Last Minute ENGINEERS, «Last Minute ENGINEERS,» 2020. [En línea]. Available: https://lastminuteengineers.com/dht11-dht22-arduino-tutorial/. [Último acceso: 16 02 2020].

[20] T. Liu, «Digital-output relative humidity & temperature sensor/module DHT22,» Co.Ltd, Aosong Electronics.

[21] F. M. Millán, Diseño e implementación de un sistema de medida de gases con Arduino, Zaragoza : Escuela Universitaria Politécnica de Teruel, 2016.

[22] L. Corona Ramírez, G. Abarca Jiménez y J. Mares Carreño, Sensores y actuadores: Aplicaciones con Arduino, México D.F.: Grupo Editorial Patria, 2014.

[23] HANWEI ELETRONICS CO. LTD, «TECHNICAL DATA MQ-2 GAS SENSOR,» HANWEI ELETRONICS CO. LTD.

[24] P. F. Jiménez Guáman y E. A. Peláez Aucay, Diseño de un Sistema de Medición y Monitoreo del Consumo de Energía por circuitos en el Hogar, Mediante Tecnología de Comunicación por Línea de Potencia, Cuenca: Universidad del Azuay, 2018.

[25] YHDC, «Split core current transformer Model：SCT-013,» YHDC, China.

[26] YHDC, «Split core current transformer Model：SCT-019,» YHDC, China.

[27] J. A. Cortés Cortés, EVALUACIÓN DE SENSORES PARA SU CONEXIÓN A TARJETA ARDUINO, Jaén: Universidad de , 2017.

[28] PCBoard.ca , «PCBoard.ca,» [En línea]. Available: https://www.pcboard.ca/5-channel-flame-detector. [Último acceso: 20 02 2020].

[29] D. V. Cruz Caiza y V. T. Martínez Ballesteros , DISEÑO Y CONSTRUCCIÓN DE UN PROTOTIPO DE SISTEMA DE ALAMCENAMIENTO Y RECUPERACIÓN AUTOMÁTICA DE PRODUCTOS, Quito: Escuela Politécnica Nácional, 2008.

[30] E. V. Altamirano Santillán, G. E. Vallejo Vallejo y J. C. Cruz Hurtado, «Diseño De vibrómetro con ArDuino y simulink,» *DIALNET INTERFASES,* nº 9, p. 22, 2016.

[31] G. Cevallos, Escritor, *Sistemas de Cableado Estructurado Parte lll.* [Performance]. 2015.

[32] Y. E. Pilco Ramos y G. G. Ovaco Sandoya, “MIDDLEWARE”, Guayaquil: ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL, 2014.

[33] M. Rouse, «TECHTARGET NETWORK,» 2019. [En línea]. Available: https://searchapparchitecture.techtarget.com/definition/RESTful-API. [Último acceso: 29 03 2020].

[34] «Introducción al mundo de las APIS,» *BBVAOpen4U,* pp. 1, 2.

[35] ca technologies, «Una guía para el diseño de API y REST,» *ca technologies.*

[36] K. Lange, «REST,» *THE LITTLE BOOK ON REST SERVICES,* 2016.

[37] Raspberrypi.org, «Raspberrypi.org,» 2020, [En línea]. Available: https://www.raspberrypi.org/. [Último acceso: 29 03 2020].

[38] A. Oppel y R. Sheldon, Fundamentos de SQL, Santa Fe: McGRAW-HILL INTERAMERICANA EDITORES, S.A. DE C.V., 2010.

[39] R. Camps Paré, L. A. Casillas Santillán, D. Costal Costa, M. Gibert Ginestà, C. Martín Escofet y O. Pérez Mora, Software libre Bases de datos, Barcelona: Fundació per a la Universitat Oberta de Catalunya, 2005.

[40] Rukbottoland y W. Warby, «Rukbottoland,» 06 04 2018. [En línea]. Available: https://rukbottoland.com/blog/tutorial-de-python-virtualenv/. [Último acceso: 30 03 2020].

[41] A. Fernández Montoro, Python 3 al descubierto, Madrid: RC Libros, 2012.

[42] pypi.org, «Flask 1.1.2,» [En línea]. Available: https://pypi.org/project/Flask/. [Último acceso: 15 03 2020].

[43] pypi.org, «Flask-HTTPAuth,» [En línea]. Available: https://flask-httpauth.readthedocs.io/en/latest/. [Último acceso: 01 04 2020].

[44] pypi.org, «Flask-RESTful,» [En línea]. Available: https://flask-restful.readthedocs.io/en/latest/. [Último acceso: 01 04 2020].

[45] pypi.org, «Flask-SQLAlchemy,» [En línea]. Available: https://flask-sqlalchemy.palletsprojects.com/en/2.x/. [Último acceso: 01 04 2020].

[46] pypi.org, «https://flask-marshmallow.readthedocs.io/en/latest/,» [En línea]. Available: https://flask-marshmallow.readthedocs.io/en/latest/. [Último acceso: 01 04 2020].

[47] pypi, «marshmallow: simplified object serialization,» [En línea]. Available: https://marshreadthedocs.io/en/stable/mallow.. [Último acceso: 01 04 2020].

[48] SQLAlchemy.org, «SQLAlchemy,» [En línea]. Available: https://www.sqlalchemy.org/. [Último acceso: 01 04 2020].

[49] pypi, «marshmallow-SQLalchemy,» [En línea]. Available: https://marshmallow-sqlalchemy.readthedocs.io/en/latest/. [Último acceso: 01 04 2020].

[50] E. S. MURCIA PEREZ y J. C. MELENDEZ MARTINEZ, MODULO WEB FRONT-END PARA EL DESARROLLO DE SIMULACION A PARTIR DE WEIBULL, JI CUADRADO Y BETA., Bogota: UNIVERSIDAD CATOLICA DE COLOMBIA, 2013.

[51] Y. Fernández Romero y Y. Díaz González, «Patrón Modelo-Vista-Controlador,» *Revista Telem@tica,* vol. 11, nº 1, pp. 47 - 49, 2012.

[52] Bootstrap team , «Bootstrap,» [En línea]. Available: https://getbootstrap.com/. [Último acceso: 01 04 2020].

[53] PUBLICACIONES VÉRTICE S.L., «Introducción al Diseño de páginas web,» de *DISEÑO BÁSICO DE PÁGINAS WEB EN HTML*, Málaga, Ediciones VÉRTICE , pp. 13, 21.

[54] AdminLTE, «AdminLTE,» [En línea]. Available: https://adminlte.io/themes/AdminLTE/documentation/index.html. [Último acceso: 02 04 2020].

[55] POSTMAN, «POSTMAN,» [En línea]. Available: https://www.postman.com/. [Último acceso: 13 04 2020].

[56] sqlitebrowser.org, «SQLITEBROWSER,» [En línea]. Available: https://sqlitebrowser.org/. [Último acceso: 29 04 2020].

[57] N. M. SAC, «Naylamp Mechatronics SAC.,» [En línea]. Available: https://naylampmechatronics.com/blog/51_tutorial-sensor-de-corriente-ac-no-invasivo-s.html. [Último acceso: 10 10 2020].