MANUAL TÉCNICO

Proyecto: Sistema de Detección de Incendios

Contenido

[RECURSOS DE HARDWARE 1](#_Toc536473537)

[RECURSOS DE SOFTWARE 1](#_Toc536473538)

[IMPLEMENTACIÓN DEL PROYECTO 2](#_Toc536473544)

[DIAGRAMAS DE DISEÑO DEL PROYECTO 4](#_Toc536473544)

[**Diagrama de Entidad-Relación** 4](#_Toc536473545)

[**Diagrama de Circuito** 5](#_Toc536473546)

[**Diagrama de Despliegue** 5](#_Toc536473547)

[ESPECIFICACIONES DE LA BASE DE DATOS 6](#_Toc536473544)

[**Entidad Ciudad** 6](#_Toc536473545)

[**Entidad Parroquia** 6](#_Toc536473546)

[**Entidad Sector** 6](#_Toc536473547)

[**Entidad Nodo** 7](#_Toc536473547)

[**Entidad Fecha** 7](#_Toc536473547)

[**Entidad Hora** 7](#_Toc536473547)

[**Entidad Sensado** 7](#_Toc536473547)

[CÓDIGO FUENTE 8](#_Toc536473544)

[**Script para Cliente** 8](#_Toc536473545)

[**Script para Servidor** 11](#_Toc536473546)

[**Script para Administración de Base de Datos** 15](#_Toc536473547)

**RECURSOS DE HARDWARE**

Para un mejor desempeño del sistema, se recomienda una PC (máquina anfitriona) con las siguientes características:

* Procesador: Core i7 CPU
* Memoria RAM: 16 GB
* Espacio en memoria (SSD o HDD): 100 GB
* Dispositivos de E/S: mouse y teclado
* Tarjeta de red

La robustez de la máquina servidor es de vital importancia para que pueda soportar las grandes cargas de procesamiento que resultan de las múltiples conexiones concurrentes con los clientes. Para el servidor, se puede instalar CentOS 7 en una partición adicional del disco y usar el gestor de arranque múltiple GRUB para bootear, o usar una máquina virtual con características potentes; lo más recomendable es la primera opción para aprovechar todo el potencial de la memoria RAM.

Por otro lado, cada nodo sensor está constituido por una plataforma de desarrollo Raspberry Pi 2 B, la cual genera tiene la tarea de sensar los niveles de temperatura y de gas en el ambiente circundante, valiéndose de un sensor DHT22 (de temperatura) y un sensor MQ7 (de gas) conectados directamente a los pines GPIO de la Raspberry; así como de enviar dichos datos a través de una conexión usando sockets hacia el servidor (que se encuentra dentro de la misma red) CentOS 7. La Raspberry empleada tiene las siguientes características:

* Procesador: Quad-core ARM Cortex-A7 CPU 900 MHz
* Memoria RAM: 1 GB
* Dispositivos de E/S: mouse y teclado
* 40 pines de entrada y salida de propósito general

**RECURSOS DE SOFTWARE**

* Oracle VM VirtualBox
* Distribución CentOS 7
* Distribución Raspbian
* Raspberry Pi 2 B
* MySQL Community Server
* Intérprete de Python para bash de Linux
* Scripts codificados en lenguaje de programación Python
* Sentencias de MySQL

**IMPLEMENTACIÓN DEL PROYECTO**

Como fase preliminar de la implementación física, la propuesta del proyecto fue implementada de forma simulada: utilizando VirtualBox, se creó una máquina virtual CentOS 7 (servidor) y una máquina virtual con SO Raspbian (cliente). El primer aspecto que se trató fue el de establecer una conexión confiable y directa para el envío y recepción de datos entre el servidor y cliente. Se optó por usar la librería ‘socket’ de Python, la cual tiene la característica de ser una especie de conversión de las librerías de socket de sistemas UNIX hacia el estilo de orientación a objetos propio de Python; de esta forma la implementación resulta más sencilla dado que la asignación de tamaños de buffer y las operaciones de recepción son automáticas.

Luego se procedió a codificar la conexión entre cliente-servidor con el intérprete de Python en Linux, llamándolo a través de la línea ‘#!/usr/bin/env python’. En el script de cliente se definieron la IP y puerto destino (IP del servidor), un identificador de nodo, y la asignación de pines para que los datos puedan ser leídos correctamente usando los sensores; además se emplearon funciones típicas de la librería ‘socket’ como ‘connect()’ y ‘send()’, con el fin de conectarse al servidor y de enviar un mensaje respectivamente. Cabe destacar que antes de la implementación física del cliente (Raspberry con sensores), se generaron valores aleatorios entre rangos típicos de temperatura y nivel de gas sensado, los cuales fueron enviados al servidor.

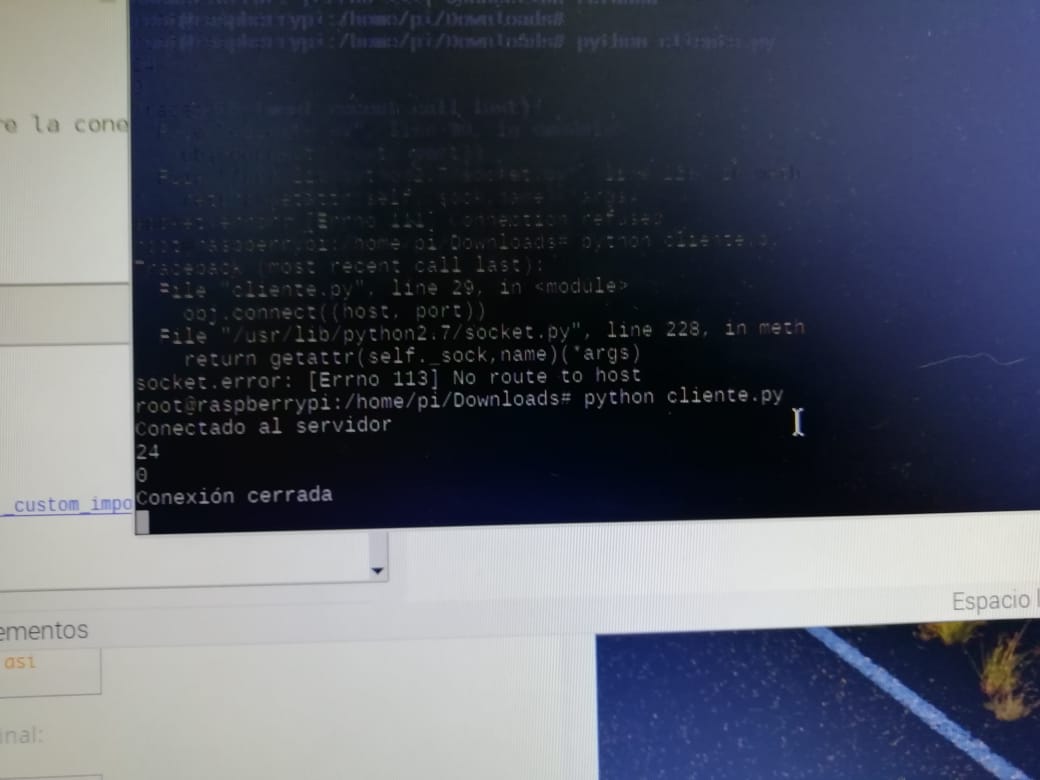
Por otro lado, en el script de servidor se importaron algunas librerías, entre ellas destacan: ‘socket’ para la comunicación con el cliente, ‘MySQLdb’ para establecer la conexión con la base de datos de MySQL, y ‘easygui’ para mostrar un mensaje de alerta a través de una ventana pequeña indicando que cierto nodo ha detectado un incendio. En el mismo script se definió un objeto conector de la base de datos con el método ‘MySQLdb.connect()’, pasándole como parámetros el host, el usuario, su contraseña y la base de datos a manejar; también se declaró un cursor con el método ‘cursor()’ con el fin de ejecutar diversas acciones propias de MySQL sobre la base de datos especificada. En adición, el servidor valida si se suscita un incendio basándose en comparación con los valores sensados respecto a valores umbrales típicos de un incidente de dicha naturaleza, y de ser el caso muestra un mensaje de alerta.

Tanto en el servidor como en cada cliente, los procesos que ocurren gracias a los scripts de ‘servidor.py’ y ‘cliente.py’ transcurren en segundo plano; después de haber ejecutado los scripts sea en el cliente o en el servidor, la comunicación perdura, y paralelamente, en el servidor se ejecuta un nuevo script: ‘menu.py’. Con este script, se logró administrador de forma más eficiente la base de datos a través de un menú en la consola de comandos, permitiendo que el administrador del servidor ingrese un nuevo nodo, elimine un sector, o genere gráficos estadísticos que faciliten el análisis de incidentes reportados por los nodos sensores (por ejemplo, un diagrama de pastel) y que a la vez ayuden a tomar las medidas pertinentes que logren disminuir los incidentes.

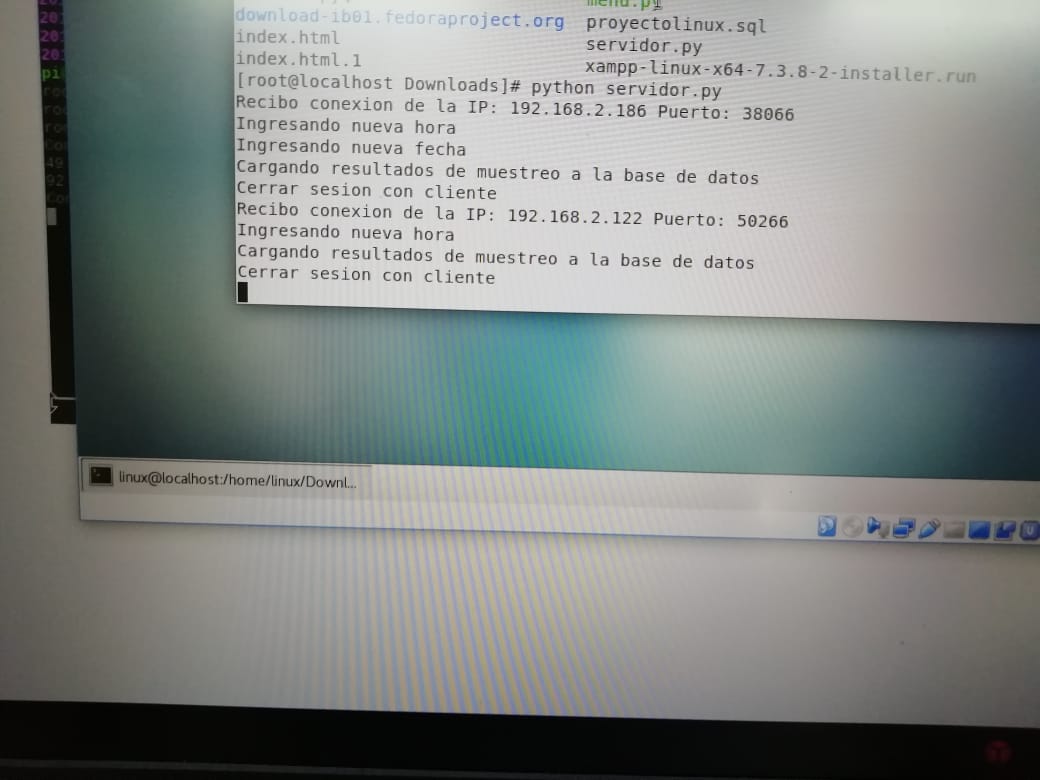
Una vez que se obtuvieron resultados correctos en la etapa de simulación, se pasó a la etapa de implementación en físico; en esta instancia solo se usó una máquina virtual con SO CentOS 7, y a través de una red local se conectó dicha máquina virtual con una tarjeta Raspberry Pi 2 B. Se cargó el script de cliente en la Raspberry y el mismo script de servidor en la máquina con CentOS 7, se configuraron las IPs de tal forma que el cliente y el servidor se vean entre sí, validando con la herramienta de ping, y se conectaron los sensores DHT22 (de temperatura) y MQ7 (de gas) a la Raspberry. Como resultado se obtuvo una correcta lectura de datos, satisfactorio envío de data sensada hacia el servidor CentOS y en dicho servidor la inserción de datos en la base de datos MySQL.



*Ilustración 1. Conexión de Raspberry a sensores y a red local*



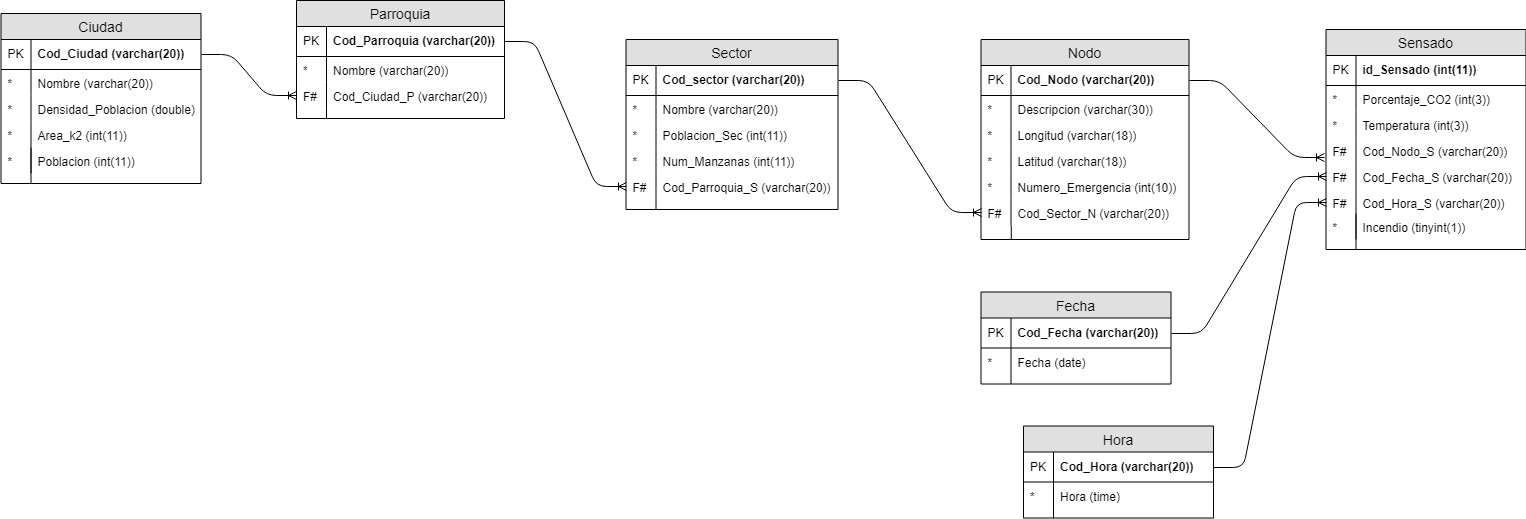
*Ilustración 2. Conexión exitosa con el servidor (la temperatura es de 24 °C y el nivel de gas 0)*



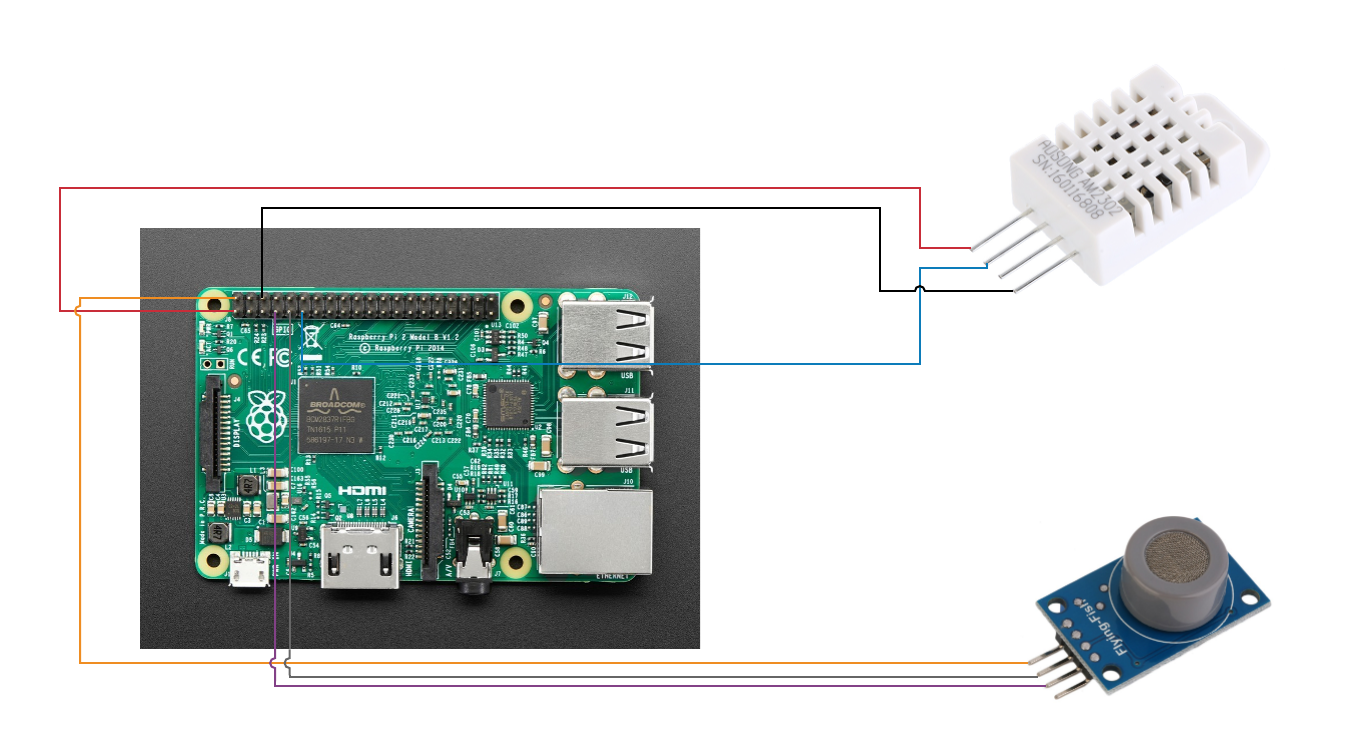
*Ilustración 3. Resultados de conexión satisfactorio con cliente*

**DIAGRAMAS DE DISEÑO DEL PROYECTO**

Diagrama de Modelo Entidad-Relación

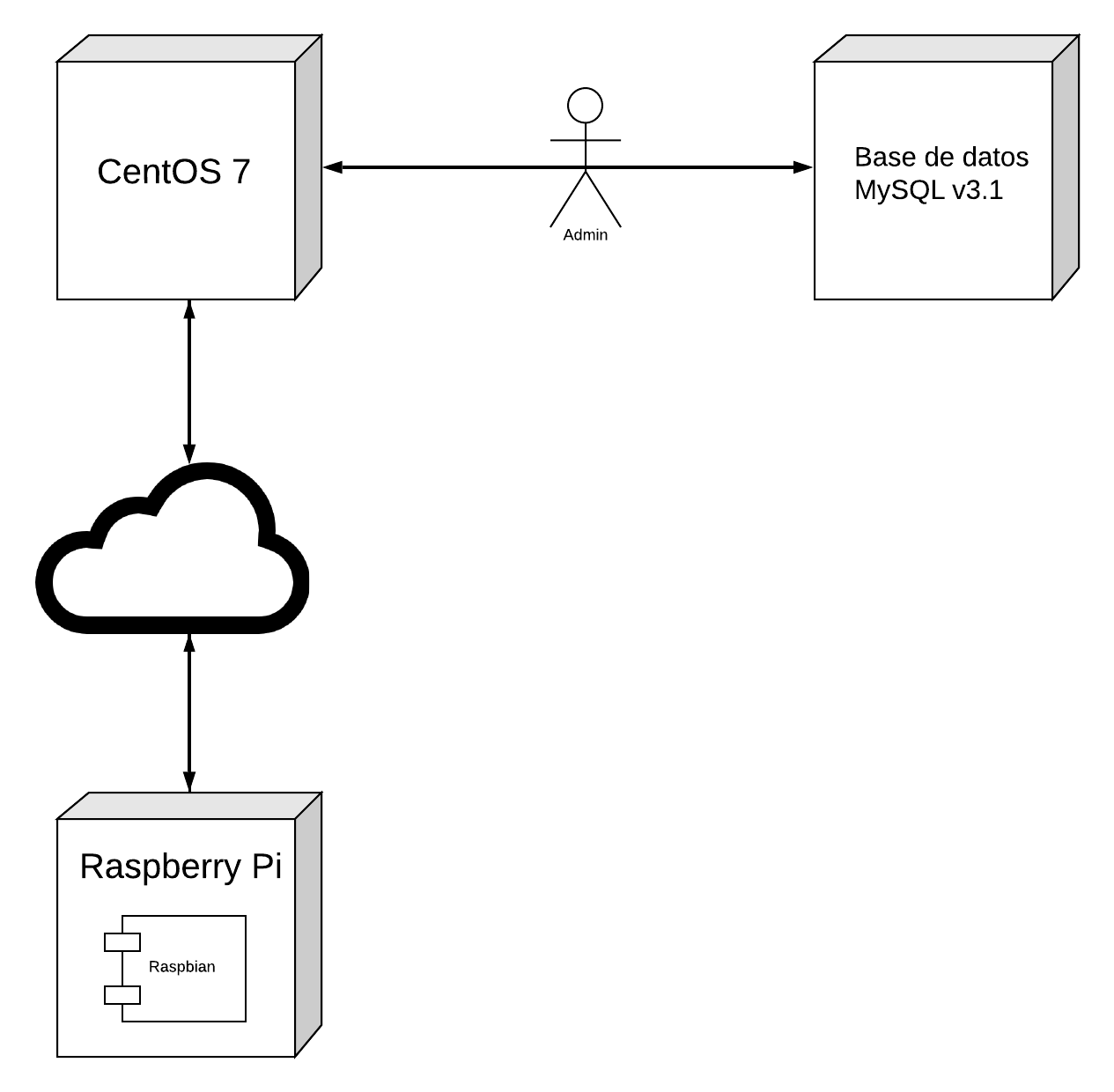


*Ilustración 4. Diagrama entidad-relación*

****Diagrama de Circuito

*Ilustración 5. Diagrama circuital*

Diagrama de Despliegue



*Ilustración 6. Diagrama de despliegue*

**ESPECIFICACIONES DE LA BASE DE DATOS**

Entidad ‘Ciudad’:

* Cod\_Ciudad: Clave primaria de la tabla ‘Ciudad’, es un identificador que caracteriza una ciudad. El tipo de dato es varchar y su longitud es de 20. Esta variable se encuentra indexada con una variable de la entidad ‘Parroquia’.
* Nombre: Atributo de carácter mandatorio, almacena el nombre de la ciudad; el tipo de dato es varchar y la longitud de 20.
* Densidad\_Poblacion: Variable obligatoria que ilustra la densidad poblacional de la ciudad; es de tipo double.
* Area\_k2: Área que comprende la ciudad en cuestión en km2, es de tipo entero y tiene longitud de 11.
* Poblacion: Cantidad de habitantes de la ciudad, variable de tipo entero y de longitud 11.

Entidad ‘Parroquia’:

* Cod\_Parroquia: Clave primaria de la tabla ‘Parroquia’, es un identificador que distingue una parroquia. El tipo de dato es varchar y su longitud es de 20. Esta variable se encuentra indexada con una variable de la entidad ‘Sector’.
* Nombre: Variable obligatoria que guarda el nombre de la parroquia, dicho nombre podrá tener como máximo 20 caracteres y es de tipo varchar.
* Cod\_Ciudad\_P: Clave foránea proveniente de la tabla ‘Ciudad’; variable que indica la ciudad en donde se encuentra la parroquia en cuestión.

Entidad ‘Sector’:

* Cod\_sector: Clave primaria de la tabla ‘Sector’, variable única que asigna un identificador a un determinado sector. El tipo de dato es varchar y su longitud es de 20. Esta variable se encuentra indexada con una variable de la entidad ‘Nodo’.
* Nombre: Atributo que guarda el nombre de un sector particular; el mismo tiene una longitud de 20 caracteres y es de tipo varchar.
* Poblacion\_Sec: Cantidad de habitantes que residen en cierto sector. La variable es de tipo entero y su longitud de 11.
* Num\_Manzanas: Número de manzanas que constituyen el sector. La variable tiene longitud de 11 y es de tipo entero.
* Cod\_Parroquia\_S: Clave foránea que es importada de la entidad ‘Parroquia’; indica la parroquia que contiene al sector, es de tipo varchar y su longitud es de 20.

Entidad ‘Nodo’:

* Cod\_Nodo: Clave primaria de la tabla ‘Nodo’, variable que no se puede repetir y que caracteriza a un nodo sensor. El tipo de dato es varchar y su longitud es de 20. Esta variable se encuentra indexada con una variable de la entidad ‘Sensado’.
* Descripcion: Breve descripción del nodo para mejor identificación del mismo; es de tipo varchar y tiene un tamaño de 30 caracteres.
* Longitud: Variable típica de ubicación geográfica de un nodo sensor. Es de tipo varchar y tiene una longitud de 18.
* Latitud: Variable típica de ubicación geográfica de un nodo sensor. Es de tipo varchar y tiene un tamaño de 18 caracteres.
* Numero\_Emergencia: Número al cual una persona se puede referir para reportar una emergencia. Es de tipo entero y tiene longitud de 10.
* Cod\_Sector\_N: Clave primaria foránea que se importa de la tabla de ‘Sector’; esta variable permite identificar de forma sencillo a qué sector pertenece el nodo en cuestión, es de tipo varchar y su longitud es de 20.

Entidad ‘Fecha’:

* Cod\_Fecha: Clave primaria de la tabla ‘Fecha’; es un identificador de una fecha en particular. Tiene longitud de 20 caracteres y es de tipo varchar.
* Fecha: Atributo de tipo date, el mismo almacena una fecha con el formato YYYY-MM-DD. Cabe destacar que en base a esta variable se generará un gráfico de barras para visualizar incidentes por fecha y por hora (esto último por una variable de la tabla ‘Hora’).

Entidad ‘Hora’:

* Cod\_Fecha: Clave primaria de la tabla ‘Hora’; es un identificador de una hora en particular. Tiene longitud de 20 caracteres y es de tipo varchar.
* Hora: Atributo de tipo time, el mismo almacena una hora con el formato hh:mm:ss (horas:minutos:segundos). Cabe destacar que en base a esta variable se generará un gráfico de barras para visualizar incidentes por hora y por fecha (esto último por una variable de la tabla ‘Fecha’).

Entidad ‘Sensado’:

* Id\_Sensado: Clave primaria de la tabla ‘Sensado’; es un identificador que debe ser único y permite identificar un sensado en particular. Tiene longitud de 11 números enteros.
* Porcentaje\_CO2: Nivel de gas detectado por el sensor MQ7; tiene longitud de 3 y es de tipo entero.
* Temperatura: Temperatura extraída por el sensor DHT22; tiene longitud de 3 y es de tipo entero.
* Cod\_Nodo\_S: Clave foránea importada de la tabla ‘Nodo’, la cual hace posible que se identifique qué nodo recolectó el valor sensado. Es de tipo varchar y su longitud es de 20.
* Cod\_Fecha\_S: Clave foránea que tiene su origen en la entidad ‘Fecha’; es la fecha en la cual se obtuvo el valor sensado en cuestión. El tipo es varchar y su longitud es de 20.
* Cod\_Hora\_S: Clave foránea proveniente de la entidad ‘Hora’; es la hora en la cual se registró un dato sensado en específico. El tipo es varchar y su longitud de 20.
* Incendio: Atributo de vital importancia que cambia si ocurre un incendio en un área determinada. Por defecto su valor es de 0, pero si los datos sensados se exceden de valores umbrales, esta variable cambia y toma el valor de 1. El tipo de dato es tinyint y su longitud es de 1.

Las sentencias SQL usadas para la consulta e inserción de datos en la base de datos de MySQL fueron usadas en el script de menú para el administrador de servidor (menu.py), personalizándolas para que el manejo de la base de datos se simplifique. Por ello dichas sentencias se encuentran en la siguiente sección, que corresponde al código fuente de todo el proyecto.

**CÓDIGO FUENTE**

Script para cliente (nodo Raspberry) ‘cliente.py’

#!/usr/bin/env python

#coding=utf-8

#Variables

#ip del servidor

host = '192.168.2.194'

port = 8085

#Se importan los módulos necesarios

import time

#from random import randint

import socket

import Adafruit\_DHT

import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM) #Ajustamos la placa en modo BCM

GPIO.setup(4, GPIO.IN) #Declaramos que el pin 4 será de entrada

nombre\_nodo=b"Alb\_Rasp1\_Guay;"

while True:

#Lectura de temperatura del sensor

humidity, temperature = Adafruit\_DHT.read\_retry(22, 17)#lectura del sensor DHT22 Conectado al pin 17

lectura = GPIO.input(4)

temp = int(temperature)

gas = int(lectura)

try:

#Creación de un objeto socket (del lado del cliente)

obj = socket.socket()

#Conexión con el servidor. Parametros: IP (puede ser del tipo 192.168.1.1 o localhost), Puerto

obj.connect((host, port))

print("Conectado al servidor")

#Convertimos a string los valores e imprimimos su valor

dato\_Temperatura = str(temp)

print(dato\_Temperatura)

dato\_Gas = str(gas)

print(dato\_Gas)

#Enviar los datos al servidor

obj.send(nombre\_nodo+dato\_Temperatura.encode()+b";"+dato\_Gas.encode())

#Cerramos la instancia del objeto servidor

obj.close()

#Imprimimos la palabra Adiós para cuando se cierre la conexión

print("Conexión cerrada")

except:

print("Error al conectarse con el servidor")

time.sleep(600)

#mesaje de fin del programa

print("Fin del programa")

El script para los clientes, ‘cliente.py’, declara variables generales de suma importancia para realizar una conexión exitosa a través de la librería de sockets de Python: la IP del servidor (host) y el puerto virtual asignado (port). Además, se importan las librerías: ‘time’ que permite esperar un tiempo prudencial para que los mensajes en consola sean más legibles para el cliente o para realizar ciertas acciones de control (bucles), ‘socket’ la cual provee mejor abstracción de las librerías de socket de UNIX, ‘Adafruit\_DHT’ para programar la lectura del sensor DHT22 y ‘RPi.GPIO’ para la detección de datos usando el sensor de gas. A continuación, con la función ‘setmode()’ se cambia a la notación de BCM, dejando de tomar en cuenta la enumeración típica de pines para basarse en la enumeración de GPIO. Con la función de ‘setup()’ se asigna el pin GPI4 para la lectura de lo sensado por el MQ7. Más adelante, se especifica un identificar/nombre para el nodo, incluyen el sector en donde se encuentra desplegado, el código de nodo y la ciudad en donde se encuentra.

Para la sección de lectura y envío de datos hacia el servidor, se construyó un bucle while que se ejecutará indefinidamente, siendo retardado cada tanto tiempo, el cual se puede configurar con la función ‘sleep()’. Dentro del lazo, se especifica que se lea la temperatura en el pin GPI17 (originalmente pin 11), y el nivel de gas en el pin GPI4 (originalmente pin 7), y se castean los valores a tipo entero. En la siguiente línea, se ingresa a un bloque try-except para manejar errores; aquí se crea un objeto de tipo socket y se intenta establecer conexión con el servidor mediante el método ‘connect()’, pasándole como parámetros la IP del servidor y el puerto asociado, si se genera un error de conexión se imprime un mensaje advirtiendo al cliente que no se pudo conectar con el servidor. Enseguida, se envía la data sensada con la función ‘send()’, no sin antes castear a String las variables de temperatura y de nivel de gas. Posterior a ello, se cierra la conexión con el servidor. El proceso mencionado recientemente se repite indefinidamente durante lapsos de tiempo, siendo retrasado por un tiempo t establecido como parámetro de la función ‘sleep(t)’, función que pertenece a la librería time. De esta forma se logra que el cliente envíe cada cierto tiempo las mediciones respectivas mientras el servidor escucha constantemente las peticiones de diversos clientes.

Script para servidor (CentOS 7) ‘servidor.py’

#!/usr/bin/env python

#Se importan los módulos necesarios

import socket

import MySQLdb

import time

import easygui as eg

#instanciamos un objeto para trabajar con el socket

ser = socket.socket()

#Puerto y servidor que debe escuchar

ser.bind(("", 8085))

#Aceptamos conexiones entrantes con el método listen. Por parámetro las conexiones simultáneas.

ser.listen(100)

while True:

#Instanciamos un objeto cli (socket cliente) para recibir datos

cli, addr = ser.accept()

#Recibimos el mensaje, con el método recv recibimos datos. Por parámetro la cantidad de bytes para recibir

recibido = cli.recv(1024)

#Si se reciben datos nos muestra la IP y el mensaje recibido

print ("Recibo conexion de la IP: " + str(addr[0]) + " Puerto: " + str(addr[1]))

#Se hace split al mensaje recibido

listaRecibida = recibido.split(";")

#Ingreso de datos sensado a la base de datos

#Conectar con la base de datos

db = MySQLdb.connect(host="localhost",user="root",passwd="Valentina\_16",db="proyectolinux")

#crear un cursor

cursor = db.cursor()

#Se asigna los valores recibidos a las respectivas variables

CodNodo=listaRecibida[0].decode()

Temperatura=listaRecibida[1]

Gas=listaRecibida[2]

#Se verifica si existe el nodo registrado

my\_query="SELECT \* FROM Nodo WHERE Cod\_Nodo='"+CodNodo+"'"

cursor.execute(my\_query)

resultado=cursor.fetchone()

mens\_nodo="El nodo = "+CodNodo+" No está registrado en la base de datos"

if(resultado==None):

eg.msgbox(msg=mens\_nodo, title="Mensaje de Alerta ")

else:

#Obtener fecha de sensado

fecha=time.strftime("%y-%m-%d")

#Obtener la hora exacta

hora=time.strftime("%H:%M:%S")

#Ingresar la hora en caso de no estar ingresada en la tabla hora

my\_query="SELECT \* FROM Hora WHERE Cod\_Hora='"+hora+"'"

cursor.execute(my\_query)

resul=cursor.fetchone()

if(resul==None):

print("Ingresando nueva hora")

my\_query="INSERT INTO Hora VALUES('"+hora+"','"+hora+"')"

cursor.execute(my\_query)

db.commit()

#Ingresar la fecha en caso de no estar ingresada en la tabla fecha

my\_query="SELECT \* FROM Fecha WHERE Cod\_Fecha='"+fecha+"'"

cursor.execute(my\_query)

result=cursor.fetchone()

if(result==None):

print("Ingresando nueva fecha")

my\_query="INSERT INTO Fecha VALUES('"+fecha+"','"+fecha+"')"

cursor.execute(my\_query)

db.commit()

#Mensaje de alerta en caso de incendio

mensaje\_alerta="Se está produciendo un incendio en el nodo: "+CodNodo

Incendio=0

if(float(Gas)>12.5):

if(float(Temperatura)>60):

Incendio=1

eg.msgbox(msg=mensaje\_alerta, title="Mensaje de Alerta ")

else:

if (float(Gas)>74 and float(Temperatura)>10):

Incendio=1

eg.msgbox(msg=mensaje\_alerta, title="Mensaje de Alerta ")

#Ingresar el dato sensado a la base de datos

print("Cargando resultados de muestreo a la base de datos")

my\_query="INSERT INTO Sensado VALUES(DEFAULT,"+Gas+","+Temperatura+",'"+fecha+"','"+hora+"','"+CodNodo+"','"+str(Incendio)+"')"

cursor.execute(my\_query)

db.commit()

#Cerrar la conexión con la base de datos

db.close()

print("Cerrar sesión con cliente")

#Cerramos la instancia del socket cliente y servidor

cli.close()

ser.close()

print("Conexiones cerradas")

En el script del servidor, se manejan 2 puntos centrales: la escucha continua hacia los clientes para establecer conexiones y recibir los datos, y la lectura e inserción de dicha información en la base de datos de MySQL. Nuevamente, se importa la librería ‘socket’ para declarar un objeto que permita manejar la conexión mediante el socket; y es en este script donde se importa la librería ‘MySQLdb’, paquete que hará posible el manejo de una base de datos para leer, actualizar/modificar, agregar o eliminar datos de la misma. También cabe recalcar que se vuelve a usar la librería ‘time’, y adicionalmente se importa la librería ‘easygui’, la cual hará posible crear una ventana que muestre un mensaje de advertencia en caso de que un incendio ocurra y alertar debidamente al administrador del servidor.

Se vuelve a crear un objeto con el método ‘socket()’, y con la función ‘bind()’ se asocia un socket a un puerto en específico. Más adelante, se escuchan conexiones por parte de los clientes, y con el método ‘listen()’ se está a la espera de que un cliente solicite una conexión con el servidor; es posible pasar por parámetro (en la función ‘listen()’) la cantidad de conexiones concurrentes que el servidor puede manejar. Si ningún cliente intentara conectarse, el servidor permanecería en la función ‘listen()’, a la espera de que un cliente aparezca en el radar, y de darse el caso, pasará a la siguiente línea, en donde inicia un lazo while. Dentro del bucle, se instancia un objeto cliente y se recupera la dirección IP del mismo cliente, mientras se acepta la conexión. La función ‘recv()’ recibe los datos sensados y los almacena en un variable llamada ‘recibido’. Si se reciben datos de forma satisfactoria, se muestra la dirección IP del cliente y el puerto asociado. Los datos obtenidos por los sensores son separados por un ‘;’ y los subvalores son guardados en una lista.

Entrando más en contexto con la base de datos, posterior a la separación de datos y almacenamiento de los mismos dentro de una lista se crea la conexión con la base de datos mediante la función ‘connect()’, pasándole como parámetros el host, el usuario, la contraseña y el nombre de la base de datos. Se crea un cursor para manejar la base de datos. Luego se crean las variables que están por ingresar a la base de datos, asignándoles los valores de los elementos de la lista creado recientemente. Se procede a verificar si el nodo (identificador del cliente, fue enviado por parte del cliente) se encuentra o no registrado en la base de datos. Si no está registrado, se extrae la fecha y hora actual, y se realiza la misma verificación tanto con la fecha como con la hora: si ninguna de las 2 existe, se guardan los datos en la base de datos. Ahora, con los datos en bruto, se realiza una comparación con los valores de umbral considerados como típicos de un incendio; si el nivel de gas se pasa de 12.5 y la temperatura sobrepasa los 60 °C, la variable ‘Incendio’ toma el valor de 1 y se muestra una ventada de alerta con un objeto de la librería ‘easygui’. Después de las verificaciones anteriores, los datos sensados son cargados a la base de datos, así como la actualización del atributo ‘Incendio’ de la entidad ‘Sensado’. Finalmente se cierran las conexiones con la base de datos y con el cliente.

Script para menú de administración (CentOS 7) ‘menu.py’

#!/usr/bin/env python

#Se importan los módulos

import MySQLdb

import time

import easygui as eg

from pylab import \*

import matplotlib.pyplot as plt

import time

#Función para ingresar un nuevo nodo a la base de datos

def ingresoNodo():

#Conectar con la base de datos

try:

#Conectar con la base de datos

db = MySQLdb.connect(host="localhost",user="root",passwd="Valentina\_16",db="proyectolinux")

#crear un cursor

cursor = db.cursor()

except:

print("Error al conectar con la base de datos")

try:

#Pedido de datos para el ingreso a la

print("Ingrese un codigo de maximo 20 caracteres")

cod\_Nodo = raw\_input()

print("Ingrese una descripcion para el nodo")

Descrip\_Nodo = raw\_input()

print("Ingrese la longitud del donde esta ubicado el nodo")

Longitud\_Nodo = raw\_input()

print("Ingrese la latitud del donde esta ubicado el nodo")

Latitud\_Nodo = raw\_input()

print("Ingrese un numero de contacto para casos de emergencia")

Numero\_Nodo = raw\_input()

aux = 0

#Verificar el cod del sector

while aux==0:

print("Ingrese el codigo del sector")

Cod\_Sect\_N = raw\_input()

my\_query="SELECT \* FROM Sector WHERE Cod\_sector='"+Cod\_Sect\_N+"'"

cursor.execute(my\_query)

result\_Nodo=cursor.fetchone()

if(result\_Nodo==None):

print("Codigo del sector no existe en la base de datos")

print("Intente con las siguientes opciones: ")

print("Cod\_Sector Nombre")

my\_query="SELECT \* FROM Sector"

cursor.execute(my\_query)

for rw in cursor.fetchall():

print(rw[0]+" "+rw[1])

time.sleep(0.5)

else:

aux = 1

#Ingreso a la base de datos

my\_query="INSERT INTO Nodo VALUES('"+cod\_Nodo+"','"+Descrip\_Nodo+"','"+Longitud\_Nodo+"','"+Latitud\_Nodo+"',"+Numero\_Nodo+",'"+Cod\_Sect\_N+"')"

cursor.execute(my\_query)

db.commit()

#Cerrar la conexion con la base de datos

db.close()

print("Nodo ingresado correctamente")

except:

print("Ha ocurrido un error de registro")

#Función para eliminar un nodo

def eliminarNodo():

#Conectar con la base de datos

try:

#Conectar con la base de datos

db = MySQLdb.connect(host="localhost",user="root",passwd="Valentina\_16",db="proyectolinux")

#crear un cursor

cursor = db.cursor()

except:

print("Error al conectar con la base de datos")

try:

print("Ingrese el código del Nodo a eliminar")

cod\_Nodo = raw\_input()

my\_query="DELETE FROM Nodo WHERE Cod\_Nodo='"+cod\_Nodo+"'"

cursor.execute(my\_query)

db.commit()

#Cerrar la conexion con la base de datos

db.close()

print("Eliminacion del nodo exitoso")

except:

print("Error al eliminar el nodo")

#Funcion para ingresar un nuevo sector a la base de datos

def ingresoSector():

#Conectar con la base de datos

try:

#Conectar con la base de datos

db = MySQLdb.connect(host="localhost",user="root",passwd="Valentina\_16",db="proyectolinux")

#crear un cursor

cursor = db.cursor()

except:

print("Error al conectar con la base de datos")

try:

#Pedido de datos para el ingreso a la

print("Ingrese un código de móximo 20 caracteres")

cod\_Sector = raw\_input()

print("Ingrese el nombre del sector")

Nombre\_sector = raw\_input()

print("Ingrese la población")

Poblacion\_Sector = raw\_input()

print("Ingrese el nómero de manzanas")

Manzanas\_Sector = raw\_input()

aux = 0

#Verificar el cod de la parroquia

while aux==0:

print("Ingrese el código de la parroquia")

Cod\_Parroquia\_S = raw\_input()

my\_query="SELECT \* FROM Parroquia WHERE Cod\_Parroquia='"+Cod\_Parroquia\_S+"'"

cursor.execute(my\_query)

result\_Parroquia=cursor.fetchone()

if(result\_Parroquia==None):

print("Código de la parroquia no existe en la base de datos")

print("Intente con las siguientes opciones: ")

print("Cod\_Parroquia Nombre")

my\_query="SELECT \* FROM Parroquia"

cursor.execute(my\_query)

for rw in cursor.fetchall():

print(rw[0]+" "+rw[1])

time.sleep(0.5)

else:

aux = 1

#Ingreso a la base de datos

my\_query="INSERT INTO Sector VALUES('"+cod\_Sector+"','"+Nombre\_sector+"',"+Poblacion\_Sector+","+Manzanas\_Sector+",'"+Cod\_Parroquia\_S+"')"

cursor.execute(my\_query)

db.commit()

#Cerrar la conexión con la base de datos

db.close()

print("Sector ingresado correctamente")

except:

print("Ha ocurrido un error de registro")

#Funcion para eliminar un sector

def eliminarSector():

#Conectar con la base de datos

try:

#Conectar con la base de datos

db = MySQLdb.connect(host="localhost",user="root",passwd="Valentina\_16",db="proyectolinux")

#crear un cursor

cursor = db.cursor()

except:

print("Error al conectar con la base de datos")

try:

print("Ingrese el código del Sector a eliminar")

cod\_Sector = raw\_input()

my\_query="DELETE FROM Sector WHERE Cod\_Sector='"+cod\_Sector+"'"

cursor.execute(my\_query)

db.commit()

#Cerrar la conexión con la base de datos

db.close()

print("Eliminación del sector exitoso")

except:

print("Error al eliminar el sector")

#Funcion para ingresar una nueva parroquia a la base de datos

def ingresoParroquia():

#Conectar con la base de datos

try:

#Conectar con la base de datos

db = MySQLdb.connect(host="localhost",user="root",passwd="Valentina\_16",db="proyectolinux")

#crear un cursor

cursor = db.cursor()

except:

print("Error al conectar con la base de datos")

try:

#Pedido de datos para el ingreso a la base de datos

print("Ingrese un código de máximo 20 caracteres")

Cod\_Parroquia = raw\_input()

print("Ingrese el nombre de la parroquia")

Nombre\_Parroquia = raw\_input()

aux = 0

#Verificar el cod de la ciudad

while aux==0:

print("Ingrese el codigo de la ciudad")

Cod\_Ciudad\_P = raw\_input()

my\_query="SELECT \* FROM Ciudad WHERE Cod\_Ciudad='"+Cod\_Ciudad\_P+"'"

cursor.execute(my\_query)

result\_Nodo=cursor.fetchone()

if(result\_Nodo==None):

print("Codigo de la parroquia no existe en la base de datos")

print("Intente con las siguientes opciones: ")

print("Cod\_Ciudad Nombre")

my\_query="SELECT \* FROM Ciudad"

cursor.execute(my\_query)

for rw in cursor.fetchall():

print(rw[0]+" "+rw[1])

time.sleep(0.5)

else:

aux = 1

#Ingreso a la base de datos

my\_query="INSERT INTO Parroquia VALUES('"+Cod\_Parroquia+"','"+Nombre\_Parroquia+"','"+Cod\_Ciudad\_P+"')"

cursor.execute(my\_query)

db.commit()

#Cerrar la conexión con la base de datos

db.close()

print("Parroquia ingresada correctamente")

except:

print("Ha ocurrido un error de registro")

#Función para eliminar una parroquia

def eliminarParroquia():

#Conectar con la base de datos

try:

#Conectar con la base de datos

db = MySQLdb.connect(host="localhost",user="root",passwd="Valentina\_16",db="proyectolinux")

#crear un cursor

cursor = db.cursor()

except:

print("Error al conectar con la base de datos")

try:

print("Ingrese el código de la parroquia a eliminar")

cod\_Parroquia = raw\_input()

my\_query="DELETE FROM Parroquia WHERE Cod\_Parroquia='"+cod\_Parroquia+"'"

cursor.execute(my\_query)

db.commit()

#Cerrar la conexión con la base de datos

db.close()

print("Eliminación de la parroquia exitoso")

except:

print("Error al eliminar la parroquia")

#Función para ingresar una nueva ciudad a la base de datos

def ingresoCiudad():

#Conectar con la base de datos

try:

#Conectar con la base de datos

db = MySQLdb.connect(host="localhost",user="root",passwd="Valentina\_16",db="proyectolinux")

#crear un cursor

cursor = db.cursor()

except:

print("Error al conectar con la base de datos")

try:

#Pedido de datos para el ingreso a la base de datos

print("Ingrese un código de máximo 20 caracteres")

Cod\_Ciudad = raw\_input()

print("Ingrese el nombre")

Nombre\_Ciudad = raw\_input()

print("Ingrese la densidad de población")

Densidad\_Ciudad = raw\_input()

print("Ingrese el área en km2")

Area\_Ciudad = raw\_input()

print("Ingrese la poblacion existente")

Poblacion\_Ciudad = raw\_input()

#Ingreso a la base de datos

my\_query="INSERT INTO Ciudad VALUES('"+Cod\_Ciudad+"','"+Nombre\_Ciudad+"',"+Densidad\_Ciudad+","+Area\_Ciudad+","+Poblacion\_Ciudad+")"

print(my\_query)

cursor.execute(my\_query)

db.commit()

#Cerrar la conexión con la base de datos

db.close()

print("Ciudad ingresada correctamente")

except:

print("Ha ocurrido un error de registro")

#Función para eliminar una ciudad

def eliminarCiudad():

#Conectar con la base de datos

try:

#Conectar con la base de datos

db = MySQLdb.connect(host="localhost",user="root",passwd="Valentina\_16",db="proyectolinux")

#crear un cursor

cursor = db.cursor()

except:

print("Error al conectar con la base de datos")

try:

print("Ingrese el código de la ciudad a eliminar")

cod\_Ciudad = raw\_input()

my\_query="DELETE FROM Ciudad WHERE Cod\_Ciudad='"+cod\_Ciudad+"'"

cursor.execute(my\_query)

db.commit()

#Cerrar la conexión con la base de datos

db.close()

print("Eliminación de la ciudad exitosa")

except:

print("Error al eliminar la ciudad")

#Función para consultar los nodos

def consultaNodos():

#Conectar con la base de datos

try:

#Conectar con la base de datos

db = MySQLdb.connect(host="localhost",user="root",passwd="Valentina\_16",db="proyectolinux")

#crear un cursor

cursor = db.cursor()

except:

print("Error al conectar con la base de datos")

try:

#Consulta de nodos

print("Cod\_Nodo Descripción")

my\_query="SELECT \* FROM Nodo"

cursor.execute(my\_query)

for rw in cursor.fetchall():

print(rw[0]+" "+rw[1])

time.sleep(0.5)

except:

print("Error con la base de datos")

#Función para consulta de sectores

def consultaSectores():

#Conectar con la base de datos

try:

#Conectar con la base de datos

db = MySQLdb.connect(host="localhost",user="root",passwd="Valentina\_16",db="proyectolinux")

#crear un cursor

cursor = db.cursor()

except:

print("Error al conectar con la base de datos")

try:

#Consulta de sectores

print("Cod\_Sector Nombre")

my\_query="SELECT \* FROM Sector"

cursor.execute(my\_query)

for rw in cursor.fetchall():

print(rw[0]+" "+rw[1])

time.sleep(0.5)

except:

print("Error con la base de datos")

#Función para consultar parroquias

def consultaParroquias():

#Conectar con la base de datos

try:

#Conectar con la base de datos

db = MySQLdb.connect(host="localhost",user="root",passwd="Valentina\_16",db="proyectolinux")

#crear un cursor

cursor = db.cursor()

except:

print("Error al conectar con la base de datos")

try:

#Consulta de parroquias

print("Cod\_Parroquia Nombre")

my\_query="SELECT \* FROM Parroquia"

cursor.execute(my\_query)

for rw in cursor.fetchall():

print(rw[0]+" "+rw[1])

time.sleep(0.5)

except:

print("Error con la base de datos")

#Función para consultar ciudades

def consultaCiudades():

#Conectar con la base de datos

try:

#Conectar con la base de datos

db = MySQLdb.connect(host="localhost",user="root",passwd="Valentina\_16",db="proyectolinux")

#crear un cursor

cursor = db.cursor()

except:

print("Error al conectar con la base de datos")

try:

#Consulta de nodos

print("Cod\_Ciudad Nombre")

my\_query="SELECT \* FROM Ciudad"

cursor.execute(my\_query)

for rw in cursor.fetchall():

print(rw[0]+" "+rw[1])

time.sleep(0.5)

except:

print("Error con la base de datos")

#Función para obtener gráficas de los registros de la base de datos

def graficaPastel():

print("Esta gráfica muestra un análisis del porcentaje de incendios que han detectado los nodos")

try:

#Conectar con la base de datos

db = MySQLdb.connect(host="localhost",user="root",passwd="Valentina\_16",db="proyectolinux")

#crear un cursor

cursor = db.cursor()

#Consultar La cantidad de nodos que hay en la base de datos

my\_query="SELECT \* FROM Nodo"

cursor.execute(my\_query)

lista\_nodos=cursor.fetchall()

y=[]

etiquetas=[]

for rw in lista\_nodos:

#De cada nodo se consulta en cuantos se produjo un incendio

my\_query="SELECT \* FROM Censado WHERE Cod\_Nodo\_C='"+rw[0]+"' AND Incendio=1"

cursor.execute(my\_query)

lista\_Cens\_N=cursor.fetchall()

acu=0

for i in lista\_Cens\_N:

acu+=1

y.append(acu)

etiquetas.append(rw[0])

#Gráfico pastel

pie(y)

#xlabel('mi gráfico')

title('Gráfico Pastel de Incendio en los Nodos')

legend( (etiquetas), loc = 'upper left')

draw()

grid(True)

show()

#cerrar la conexión con la base de datos

db.close()

except:

print("Error al graficar")

#Función de grafico de barras por fechas

def graficoBarras():

try:

#Conectar con la base de datos

db = MySQLdb.connect(host="localhost",user="root",passwd="Valentina\_16",db="proyectolinux")

#crear un cursor

cursor = db.cursor()

#Pedido de fechas de consulta

print("Ingrese la fecha de inicio con formato y-m-d")

FechaInicio=raw\_input()

print("Ingrese la fecha final con formato y-m-d")

FechaFinal=raw\_input()

#Declarar las listas vacias

y=[]

etiquetas=[]

#Consulta de cuantos nodos se tiene en la base de datos

my\_query="SELECT \* FROM Nodo"

cursor.execute(my\_query)

for rw in cursor.fetchall():

#Consulta de la cantidad de incendios en rango de fechas de cierto nodo

my\_query="SELECT \* FROM Censado WHERE Cod\_Fecha\_C BETWEEN '"+FechaInicio+"' AND '"+FechaFinal+"' AND Cod\_Nodo\_C='"+rw[0]+"' AND Incendio=1"

cursor.execute(my\_query)

lista\_Censad=cursor.fetchall()

acu=0

for i in lista\_Censad:

acu+=1

y.append(acu)

etiquetas.append(rw[0])

#Gráfico de barras

fig = plt.figure(u'Grafico de barras')

ax = fig.add\_subplot(111)

xx = range(len(y))

ax.bar(xx, y, width=0.8, align='center')

ax.set\_xticks(xx)

ax.set\_xticklabels(etiquetas)

plt.show()

except:

print("Error en el proceso")

#Menú del programa

op=-1

while op!=15:

print("====================================================================")

print(" MENÚ DE ADMINISTRACIÓN DE BASE DE DATOS ")

print("====================================================================")

print("1. Ingresar un nuevo nodo")

print("2. Eliminar un nodo existente")

print("3. Ingresar un sector")

print("4. Eliminar un sector")

print("5. Ingresar una parroquia")

print("6. Eliminar una parroquia")

print("7. Ingresar una ciudad")

print("8. Eliminar una ciudad")

print("9. Gráfica pastel del total de incendios en los nodos")

print("10. Gráfico de barras de incendios por nodo segun una fecha")

print("11. Consulta de nodos registrados")

print("12. Consulta de sectores reguistrados")

print("13. Consulta de parroquias registradas")

print("14. Consulta de ciudades registradas")

print("15. Salir")

try:

print("Ingrese una opción: ")

op = input()

except:

print("Error al ingreso de opción")

if(op==1):

ingresoNodo()

time.sleep(2)

elif(op==2):

eliminarNodo()

time.sleep(2)

elif(op==3):

ingresoSector()

time.sleep(2)

elif(op==4):

eliminarSector()

time.sleep(2)

elif(op==5):

ingresoParroquia()

time.sleep(2)

elif(op==6):

eliminarParroquia()

time.sleep(2)

elif(op==7):

ingresoCiudad()

time.sleep(2)

elif(op==8):

eliminarCiudad()

time.sleep(2)

elif(op==9):

graficaPastel()

time.sleep(2)

elif(op==10):

graficoBarras()

time.sleep(2)

elif(op==11):

consultaNodos()

time.sleep(2)

elif(op==12):

consultaSectores()

time.sleep(2)

elif(op==13):

consultaParroquias()

time.sleep(2)

elif(op==14):

consultaCiudades()

time.sleep(2)

elif(op==15):

print("Gracias por su visita")

time.sleep(2)

else:

print("Opcion incorrecta")

time.sleep(2)

El script de menú (menú.py) que es ejecutado desde el servidor abarca la mayor cantidad de código, ya que el mismo permite ejecutar distintas opciones sobre la base de datos y también procesar los datos de dicha base de datos para mostrar gráficas estadísticas en base a la información recolectada por los nodos sensores. Se vuelven a emplear las librerías ‘MySQLdb’, ‘time’ y ‘easygui’, y aparecen nuevas librerías como ‘pylab’ y ‘matplolib.pyplot’ usadas para generar las gráficas mencionadas. El menú está constituido por 14 opciones (sin incluir la opción de salida del menú); la mayoría de ellas están orientadas a la administración de la base de datos MySQL. Entre ellas se encuentran las opciones para ingresar un nuevo nodo sensor, eliminar uno ya existente, ingresar o eliminar un sector, ingresar o eliminar una parroquia ya registrada, ingresar o eliminar una ciudad, mostrar una gráfica pastel y una gráfica de barras con respecto a criterios de filtrado de fecha y hora, así como realizar querys para mostrar los nodos, sectores, parroquias o ciudades existentes en la base de datos.

Para cada opción se definen procedimientos que fueron llamados dentro de una estructura condicional (para el control del menú), los cuales permiten ejecutar la opción que el administrador seleccione. En cada procedimiento codificado se abre una conexión con la base de datos, y después de haber realizado las operaciones SQL se cierra dicha conexión; de esta manera se abre la conexión únicamente cuando es necesario y se ahorra capacidad de procesamiento. Las funciones que componen el menú son:

* ingresoNodo

En primer lugar, se usa un bloque try-except en caso de que la conexión no resulte exitosa. Se procede a realizar la conexión con la base de datos. A través de la consola, se muestran los parámetros que el usuario debe ingresar, los mismos que componen la entidad ‘Nodo’ del modelo Entidad-Relación. También se solicita que ingrese la variable de código de sector, Cod\_Sector\_N, la cual es la clave foránea de la entidad ‘Nodo’. Si la clave es errónea (no existe o está mal escrita) y se muestra un mensaje de error, generando posteriormente una excepción que es manejada por el bloque lógico try-except. Nuevamente, el usuario debe intentar ingresar un código de sector apropiado; de ser el caso se ejecuta un query que devuelve un set con todos los atributos del nodo basándose en el sector. Acto seguido se ingresan las características del nodo a la base de datos, más específicamente a la tabla ‘Nodo’, se cierra la conexión con la base de datos y el procedimiento finaliza.

* eliminarNodo

Esta función realiza una conexión con la base de datos, instancia un manejador de base de datos SQL, y pide que se ingrese el código del nodo eliminar por consola. Si la consulta con la base es exitosa, se ejecuta el query, caso contrario se genera una excepción y se muestra un mensaje de error (puede ser porque el nodo no existe en la base de datos o porque el usuario cometió un error de tipografía ingresando el código del nodo). Con la sentencia de SQL se elimina únicamente el nodo especificado, sin afectar el resto de nodos. Al final se cierra la conexión con la BD y la función termina su ejecución.

* ingresoSector

Función que tiene una estructura similar a la función de ‘ingresoSector’; cuenta con un bloque try-except tanto para la conexión con la base de datos como para los querys SQL ejecutados. Se realiza y se instancia una conexión con la base de datos. Después se pide por consola los datos propios de la tabla ‘Sector’, se valida el código de la parroquia que el usuario ingresa, verificando si el mismo existe o no en la BD; si el mismo existe, se procede a insertar los datos del sector en la base de datos. Finalmente se cierra la conexión y la función cesa.

* eliminarSector

Función similar al procedimiento ‘eliminarNodo’, la cual usa 2 bloques try-except para manejar excepciones, el usuario ingresa el sector que desea retirar de la base de datos; si el sector existe y es válido se ejecuta el query y se da de baja al sector especificado. Al final del procedimiento se cierra la conexión con la BD.

* ingresoParroquia

Función que permite el ingreso de una parroquia con sus atributos asociados, los cuales son introducidos por el usuario a través de consola. Nuevamente, se manejan los errores originados por error en la conexión con la BD, o por cuestión de un query no exitoso (sea porque la variable no existe o porque está mal redactada). Si todo se realiza de forma correcta, la inserción de los datos en la BD se ejecuta con normalidad, cerrando la conexión con la base de datos posteriormente.

* eliminarParroquia

Procedimiento similar a los de eliminación de sector o de nodo, se crea un objeto de conexión, se instancia un cursor para el manejo de la BD seleccionada, y se pide al usuario que ingrese el código de parroquia que desea eliminar, mientras que en segundo plano 2 bloques de try-except proporcionan manejo de errores. Si no se generan excepciones, la parroquia en cuestión es eliminada y se cierra la conexión con la BD.

* ingresoCiudad

Este procedimiento funciona de manera similar a los de ingreso de parroquia o de sector; consta de bloques try-except que validan posibles errores que puedan ocurrir durante consultas a la base de datos o la conexión con la misma. El usuario ingresa por consola los datos característicos de la entidad ‘Ciudad’, si no existen errores la información es introducida en la base de datos en la tabla correspondiente, continuando con el proceso de cierre de conexión con la BD.

* eliminarCiudad

La función ‘eliminarCiudad()’ realiza la conexión con la BD, asigna un cursor como manejador de la misma, solicita que el usuario ingrese el código de la ciudad que desea eliminar, y valida si el query es correcto. De ser el caso, se remueve la ciudad especificada y finalmente se cierra la conexión con la base de datos.

* consultaNodos

Cambiando un poco de índole de acciones en torno a la administración de base de datos, la función en cuestión realiza únicamente tareas de consulta según sea la entidad que se requiera describir mediante un query SQL. En este caso, se muestran los atributos propios de la entidad ‘Nodo’, también empleando manejo de errores con bloques try-except.

* consultaSectores

Otra tarea de consulta de base de datos está orientada a la tabla de ‘Sector’, el procedimiento actual se conecta con la BD e intenta extraer los datos asociados a la tabla de sectores. Si no se presenta un error se ejecuta el query que permite mostrar por pantalla las variables de interés.

* consultaParroquias

Nuevamente, con el fin de permitir que el usuario tenga una visión general del estado de la base de datos sin necesidad de usar sentencias SQL, la presente función se conecta con la BD, e intenta recuperar los datos asociados a las parroquias existentes en la base de datos. De no ocurrir error alguno el query es exitoso, se muestran las variables respectivas y se finaliza la conexión con la BD.

* consultaCiudades

Finalizando con las tareas de consultas, el procedimiento en cuestión intenta consultar las variables asociadas a la entidad ‘Ciudad’ mediante la ejecución de un query SQL; se muestran los datos indexados a la tabla de ciudades si no ocurre ninguna excepción y se cierra la conexión con la base de datos.

* graficaPastel

Como aspecto adicional a la consulta, eliminación y modificación de datos de la BD, se implementaron 2 funciones que procesan la información sensada y los datos asociados a la misma, todo esto para generar gráficas intuitivas y descriptivas que permitan visualizar de forma más clara los incidentes suscitados. El procedimiento actual selecciona todos los nodos que reportaron un incendio, los agrega a una lista, y en base a los mismos muestra una gráfica pastel evidenciando qué nodo sensó mayor cantidad de incendios, dato de suma importancia para la toma de decisiones.

* graficoBarras

Siguiendo la misma idea concebida que el procedimiento anterior, en esta función se solicita el ingreso de una fecha de inicio y una de fin. En base a estas variables, se realiza una consulta a la base de datos con filtrado tomando como referencia las fechas especificadas, y se recuperan los nodos que reportaron un incidente entre este rango de fechas. Con la información obtenida, se logra generar el gráfico de barras, dejando al descubierto qué nodo estuvo inmerso en incendios con mayor frecuencia respecto de los otros. En esta función se usó la librería ‘matplotlib.pyplot’.