

Manual de instalación para sistema de domótica.

Moros Solano Pablo Andrés, Agente de Implementaciones – Sistemas de Bucaramanga.
Álvarez García Juan David, Practicante de ingeniería – Sistemas de Bucaramanga.

I. INTRODUCCIÓN

Este manual está diseñado para guiar en la implementación física del sistema, permitiendo controlar diversos dispositivos eléctricos de manera eficiente y remota en la sucursal.

La domótica es una tecnología que integra sistemas de control y automatización, mejorando la comodidad, seguridad y eficiencia energética. En este proyecto, se utilizará el microcontrolador ESP32 debido a su capacidad de conectividad Wi-Fi y Bluetooth, así como su versatilidad y potencia de procesamiento. El módulo de 4 relés permitirá controlar hasta cuatro dispositivos eléctricos diferentes, como luces, sistemas de climatización, equipos de seguridad y otros dispositivos esenciales para el funcionamiento de la sucursal.

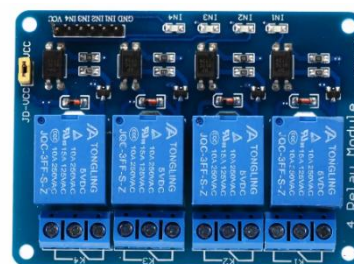
Este manual se centrará exclusivamente en la instalación del hardware necesario para el sistema de domótica, así como aspectos de configuración o programación del ESP32 y la base de datos en MySQL. El objetivo es proporcionar una guía clara y detallada para la correcta instalación de los componentes físicos del sistema.

II. OBJETIVO

- Implementar un sistema domótico eficiente y seguro en las sucursales, con el fin de optimizar el control y la gestión de los recursos energéticos, mejorar la seguridad y aumentar la comodidad tanto para los empleados como para los clientes.

III. COMPONENTES

- **ESP32:** es un chip que combina Wi-Fi y Bluetooth en un solo dispositivo. Se utiliza principalmente en proyectos de Internet de las cosas (IoT) debido a su versatilidad y capacidad de conectividad inalámbrica.
- **Módulo de 4 relés:** es un dispositivo que funciona a 5 voltios y puede manejar cargas de hasta 10 amperios a 250 voltios. Está diseñado para activar dispositivos como lámparas, ventiladores o electrodomésticos mediante señales digitales desde una placa controladora. Cada uno de los cuatro relés se controla a través de los terminales IN1, IN2, IN3 o IN4
- **Shield ESP32:** es un módulo que se conecta directamente a la placa principal para añadir funcionalidades adicionales de manera sencilla. Estas tarjetas están diseñadas para ampliar las capacidades del ESP32 sin necesidad de conexiones complicadas o soldaduras.



- **Adaptador AC/DC:** es un dispositivo que convierte la corriente alterna (AC) en corriente continua (DC). En este caso, el adaptador de 5V y 2A se utiliza principalmente para alimentar dispositivos electrónicos que requieren una fuente de alimentación estable a 5 voltios.



- **Cable USB A a USB C:** permite conectar dispositivos con puertos USB tradicionales (USB-A) a dispositivos más modernos con puertos USB-C.



- **Borneras:** son dispositivos utilizados para conectar o desconectar circuitos eléctricos de manera segura y controlada. Su función principal es repartir el suministro eléctrico a través del cableado hasta los dispositivos correspondientes.



- **Contactador:** es un interruptor electromecánico que permite controlar circuitos eléctricos de alta potencia mediante una señal de baja potencia.



A. Herramientas recomendadas

Herramientas para la implementación del sistema
Jumpers
Multímetro
Destornilladores
Pinzas
Tornillos y tuercas
Tornillos separadores

Tabla 1. Herramientas recomendadas para implementación del sistema.

V. CONEXIONES ELECTRICAS

A. Microcontrolador y relé.

Las conexiones entre el módulo de relés y la placa de expansión para ESP32 se emplean conectando los pines 33, 26, 14 y 13 de la placa con las entradas IN1, IN2, IN3 y IN4 del módulo respectivamente. De la siguiente forma:

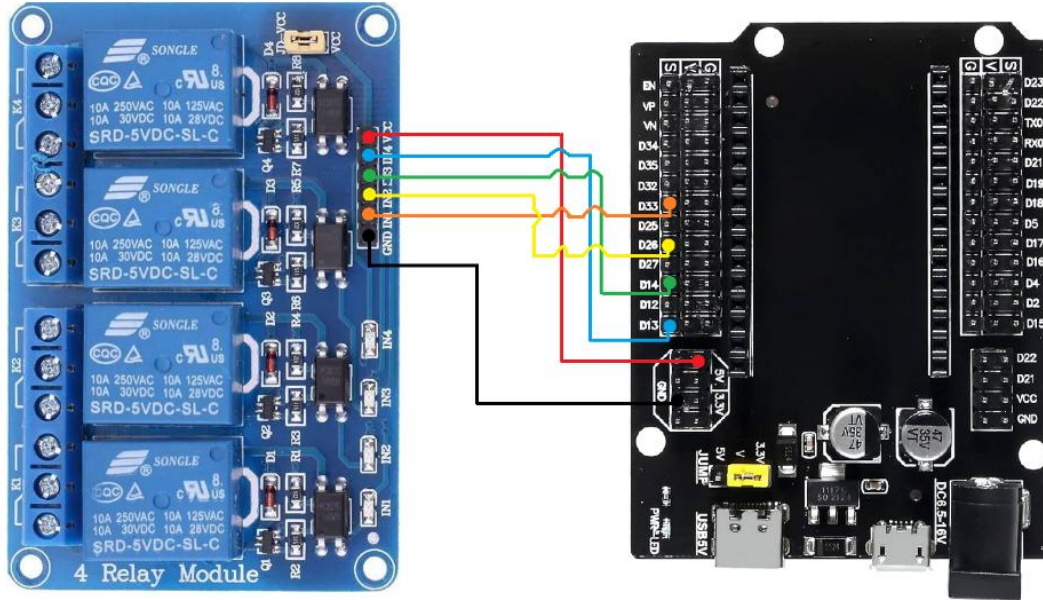


Imagen 1. Conexiones del ESP32 con el módulo de relés

Se deben conectar todos los comunes (pin central) de los 4 relés del módulo, como se observa en la imagen:

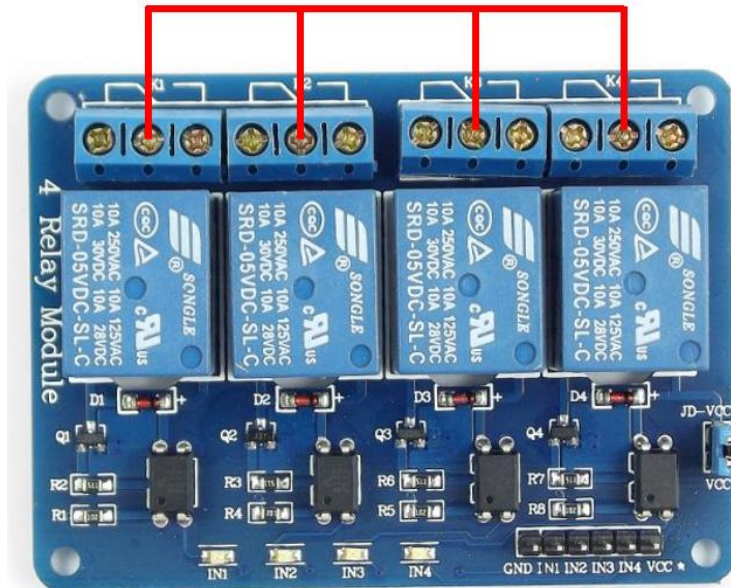


Imagen 2. Interconexión de comunes en módulo de relés.

B. Relés y contactores.

Una vez teniendo las conexiones de los dispositivos electrónicos, se debe proceder con las conexiones eléctricas entre el módulo de relés, las borneras y los contactores. El diagrama de estas conexiones se explica con la siguiente imagen:

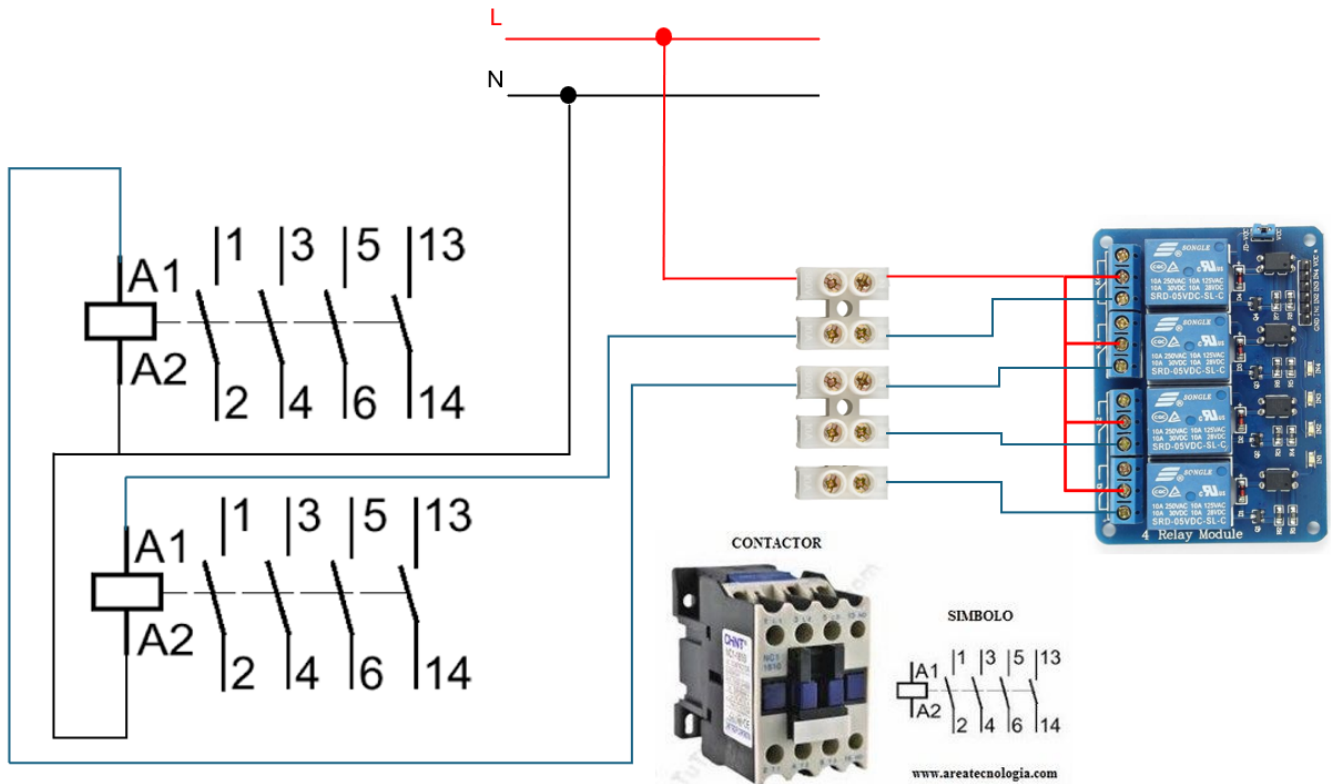


Imagen 3. Diagrama de conexión eléctrica.

Tal como se observa en el diagrama, los contactos A2 de los 4 contactores deben estar conectados al mismo nodo y la salida de las borneras deben conectarse al A1 de cada contactor. Así mismo, las cargas eléctricas deben conectarse únicamente a los pines 1-2, 3-4 y 5-6.

Debe tenerse en cuenta que cada contactor va a accionarse en un horario según se disponga en la base de datos, por lo que se le pueden conectar 3 cargas que cuenten el mismo programa de encendido y apagado a 1 mismo contactor.

VI. PROGRAMACIÓN DE ARDUINO

A. Instalación y configuración del Arduino IDE.

1) Descargar Arduino IDE.

- Abrir el siguiente link en el navegador: <https://www.arduino.cc/en/software>
- Bajar y seleccionar, en “Opciones de Descarga” o “Download Options”, el sistema operativo, que usualmente es Windows, y escoger la opción que dice “Win 10 and newer, 64 bits”. Tal como:

Downloads



Imagen 4. Opción de descarga Arduino IDE.

- Ejecutar el archivo con terminación “.exe” que se descargó. Dar clic en “Acepto”, “Siguiente” e “Instalar” para comenzar la instalación. Si aparecen mensajes solicitando permisos de administrador para instalar diferentes dependencias, dar en “Sí” o “Instalar”.

2) Configuración inicial del Arduino IDE

a) Librería ESP32.

Seleccionar “Board manager” y buscar “ESP32”, Instalar la segunda opción llamada “esp32 by Espressif”. Si se necesita permisos de administrador para instalar la dependencia, dar “Sí” o “Instalar”.

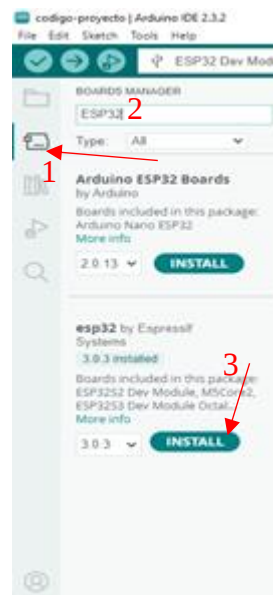


Imagen 5. Instalación librería ESP32.

b) Driver ESP32

- Ingresar al siguiente enlace:
<https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers?tab=downloads>
- Bajar a la sección “Software Downloads” y descargar la opción “CP210x Universal Windows Driver”
- Descomprimir el archivo “.zip” descargado e ingresar a la carpeta.

Nota: Esta sección sólo se debe realizar en caso de que al conectar el ESP32 y cargar el código de prueba, se evidencie un error en los puertos del computador que impida cargar correctamente el código a la placa.

- Dar clic derecho en “silabser.inf” y seleccionar la opción de “instalar”.

c) Código de prueba.

Abrir el archivo llamado “código-prueba.ino” adjunto a este manual.

Dar clic en “Tools” > “Board” > “esp32” > seleccionar “ESP32 Dev Module”. Tal como se observa en la imagen a continuación:

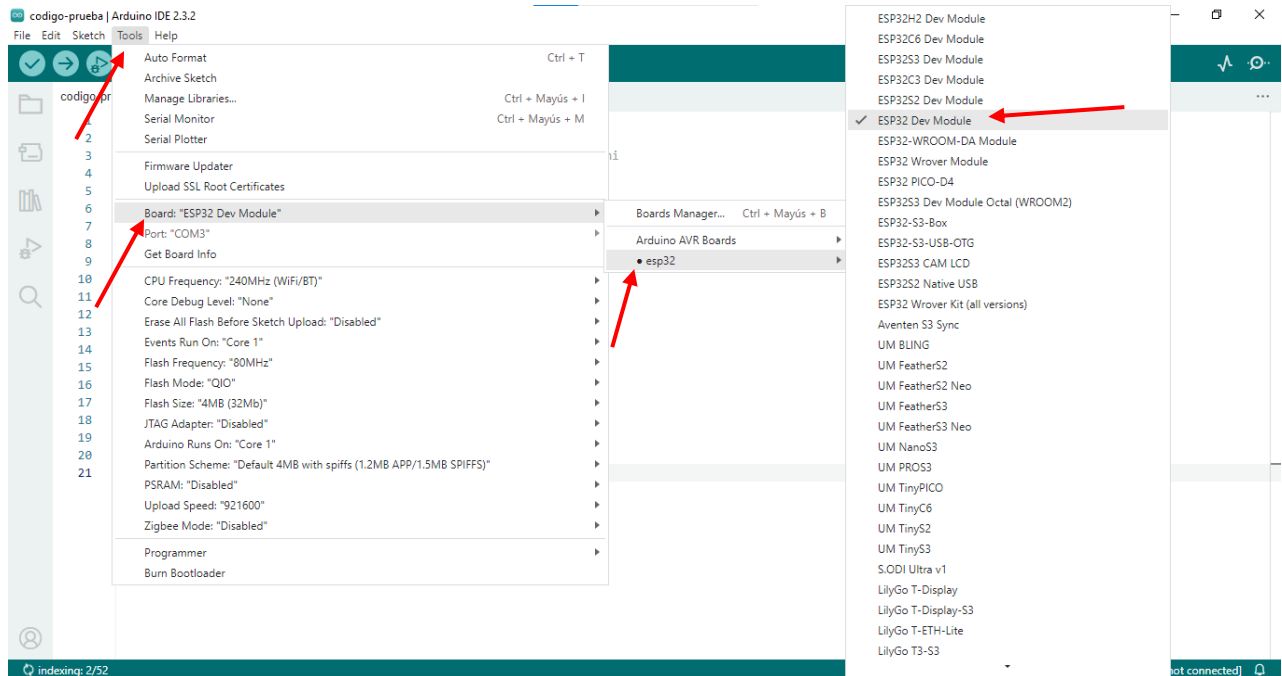


Imagen 6. Seleccionar placa.

El procedimiento es bastante similar para seleccionar el puerto. Se debe conectar el microcontrolador a través del cable conversor USB A a USB C con el computador. En caso de que se tengan varios COM# detectados, se recomienda probar puertos para saber cuál pertenece al del microcontrolador. Para poder seleccionar el puerto dentro de la interfaz del Arduino IDE se debe seguir la siguiente secuencia: *Tools > Port > "seleccionar puerto"* tal como se observa en la imagen:

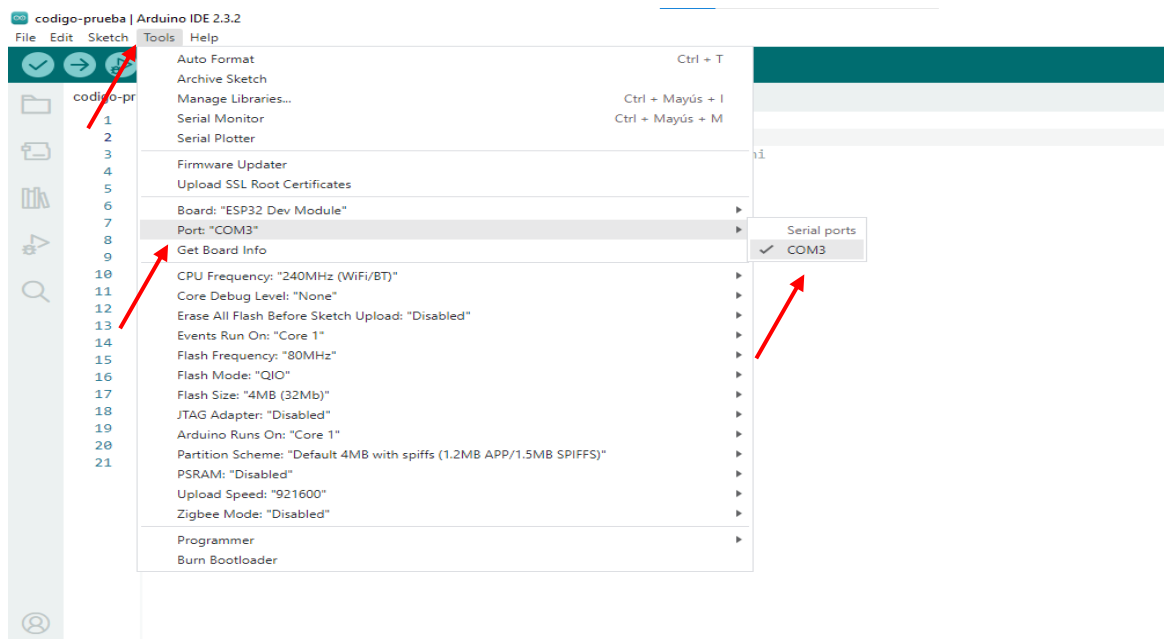


Imagen 7. Seleccionar puerto.

Una vez configurado puerto y placa, se debe ejecutar el código de prueba. Para ello, se debe dar clic en la opción que se llama “Upload” o “Cargar”.



Imagen 8. Botón de cargar.

El código iniciará a cargarse y debería poder ejecutarse sin ningún inconveniente. Una vez se haya cargado, se debe observar encender y apagar una luz led de color azul en la placa ESP32. De esta forma verificamos que la configuración de la placa fue efectuada de manera correcta y es momento de cargar el código principal del sistema.

B. Configuración del código de Arduino.

Abrir el archivo “*código-proyecto.ino*” y configurar lo siguiente:

1) IP del dispositivo.

Se debe asignar una IP al ESP32 en las líneas que se ven a continuación:

```
18  IPAddress ip(192, 168, 1, 37); // Dirección IP estática que deseas asignar al ESP32
19  IPAddress gateway(192, 168, 1, 1); // Puerta de enlace predeterminada de tu red
20  IPAddress subnet(255, 255, 255, 0); // Máscara de subred de tu red
```

Imagen 9. Configuración de IP.

2) WiFi.

Cambiar la línea 23 y 24 con el nombre y contraseña de la red WiFi donde dice “ssid” y “password” respectivamente.

```
22  // Configura los parámetros de tu red WiFi
23  const char* ssid = "Starlink";
24  const char* password = "Unidroga2023.";
```

Imagen 10. Configuración del WiFi.

Una vez configurados los aspectos anteriores, se debe cargar el archivo al dispositivo que se esté utilizando.

3) Verificación del código

Para verificar que el código se haya cargado de manera correcta, se debe abrir el monitor serial del Arduino IDE dando clic en *Tools > Serial Monitor* según:

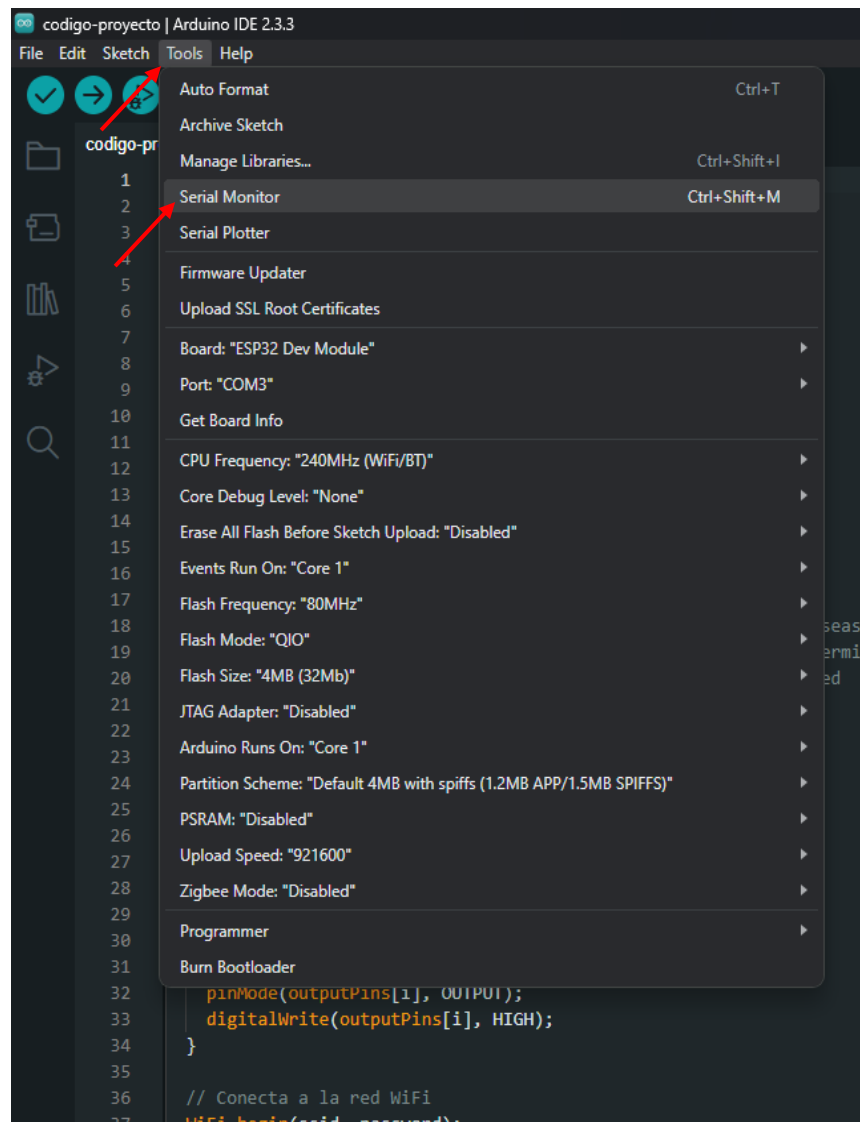


Imagen 11. Abrir serial monitor.

Posterior a ello, se debe cambiar la tasa de transmisión de datos del monitor serial a 115200 baudios, según:

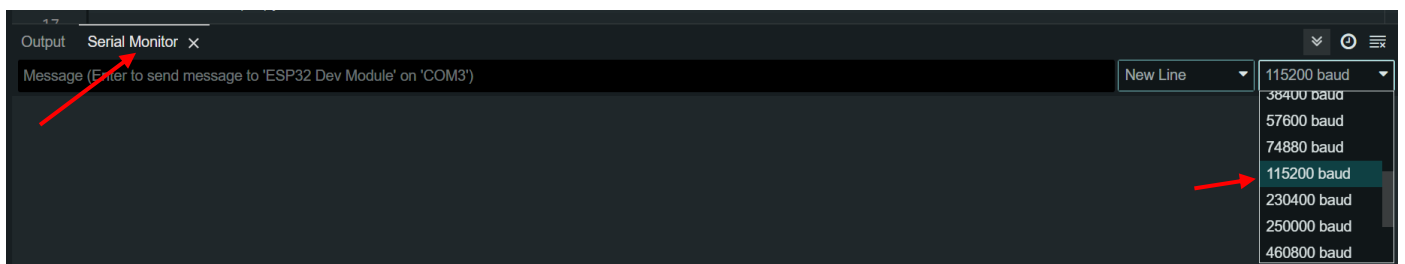


Imagen 12. Cambio de la tasa.

Finalmente, se debe dar clic en el botón de la placa ESP32 que se llama *EN*; al hacerlo, se debe observar en el monitor serial un mensaje que dice “Wi-Fi conectado” seguido de la IP que se le haya configurado al dispositivo. Así se verifica que el código y la configuración se haya realizado de manera correcta y sin ningún error.

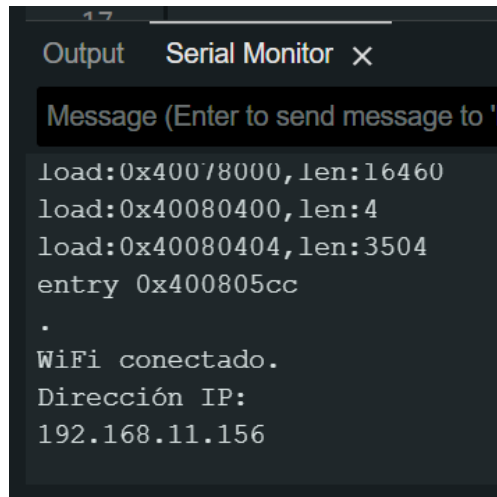


Imagen 13. Verificación de la IP.

VII. CREACIÓN DE LA BASE DE DATOS.

Esta base de datos está diseñada para gestionar un sistema domótico con cuatro tablas principales. La tabla `domotica_dispositivos`, almacena información sobre los dispositivos conectados incluyendo su identificador único y dirección IP. La tabla `domotica_onoff`, configura el estado de los pines (encendido o apagado) y establece una relación con los dispositivos a través de una clave foránea. La tabla `domotica_eventos`, permite programar eventos de encendido y apagado en horarios específicos para cada dispositivo configurado. Finalmente, la tabla `domotica_configuracion`, guarda ajustes generales del sistema como parámetros y valores de configuración.

Adjunto a este instructivo hay un archivo llamado `creación_bd.sql`, el cual debe abrirse de la siguiente forma en el POS:

1. Copiar en el pos el archivo sql que se va a ejecutar.
2. Ir al directorio en donde se encuentra el archivo.
3. Ejecutar el siguiente comando: `mysql -u root -p < creación_bd.sql`
 - a. Al hacer esto, el sistema solicitará una contraseña, la cual no se especificará acá por temas de seguridad.

Si se ejecutaron los pasos correctamente y no hubo ningún error, se debió haber creado la siguiente base de datos:

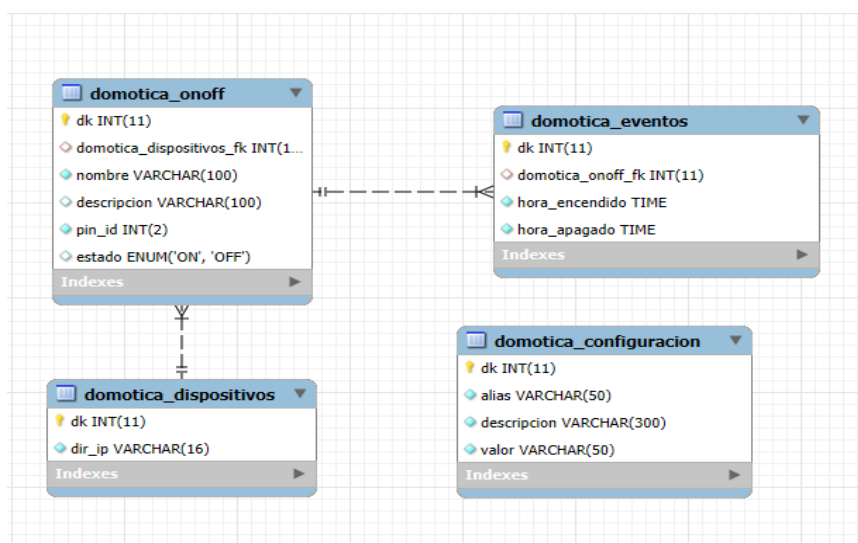


Imagen 14. Diagrama de la base de datos.

A. Ingreso de dispositivos y estados iniciales.

1) Insert en la tabla de dispositivos.

Para ingresar nuevos dispositivos en la base de datos se utiliza el siguiente insert:

```
insert into domotica.domotica_dispositivos(dir_ip)
values ('192.168.11.155');
```

La dirección IP va a variar en función de la que se haya especificado en el código de Arduino. De esta forma, si se tienen varios dispositivos, éstos deben tener, IP's diferentes y deben ingresarse a la base de datos usando el insert anterior.

2) Asignación de pines y estados iniciales.

Para configurar cada pin de cada dispositivo se utiliza el siguiente insert:

```
insert into domotica.domotica_onoff(
    domotica_dispositivos_fk,
    nombre,
    descripcion,
    pin_id,
    estado
) values
(1, 'Pin prueba', 'Es el led que trae cada dispositivo', 0, 'ON');
```

El valor que se indica en domotica_dispositivos_fk es el identificador único del dispositivo que asigna la tabla domotica_dispositivos cada vez que se ingresa uno nuevo. Los valores de pin_id van de 0 a 4, siendo cero **SIEMPRE** el pin del led interno de cada dispositivo, a partir de 1 se puede asignar a cualquier línea que se desee controlar, ya sean góndolas, luces, avisos, etc. El estado debe ser ON únicamente para el pin_id = 0, para el resto se puede escoger entre especificarlo (ON u OFF) o no, dado que por defecto se crea en ON.

B. Ingresar eventos.

Para ingresar un nuevo evento, es decir, determinar los horarios de encendido y apagado de un elemento en específico durante el día se usa el siguiente insert:

```
INSERT INTO domotica.domotica_eventos (domotica_onoff_fk, hora_encendido, hora_apagado)
VALUES (1, '00:00:00', '23:59:00');
```

El valor de domotica_onoff_fk lo determina el identificador único que asigna la tabla domotica_onoff sobre un pin de cualquier dispositivo. Los horarios se deben ingresar en formato de 24 horas según **HH:MM:SS**. El funcionamiento radica en que, durante el tiempo que se especifica, el elemento que se desee controlar que esté asignado al pin del identificador único permanecerá encendido.

VIII. PROGRAMACION PYTHON

A. Instalación de las librerías

El archivo adjunto llamado “codigo_principal.py” contiene el código que realiza todas las consultas y updates de la base de datos, así como la comunicación con el dispositivo ESP32 usado. Para su correcto funcionamiento es necesario instalar 2 librerías en el POS:

Librería	Instalación
mysql.connector	pip install mysql-connector-python
requests	pip install requests

Tabla 2. Comandos para instalar librerías

Nota: En caso de que el sistema no cuente con el gestor de paquetes PIP, se puede instalar ejecutando el archivo adjunto llamado “get-pip.py”. Igualmente es importante corroborar las versiones tanto de pip como de python usando --versión.

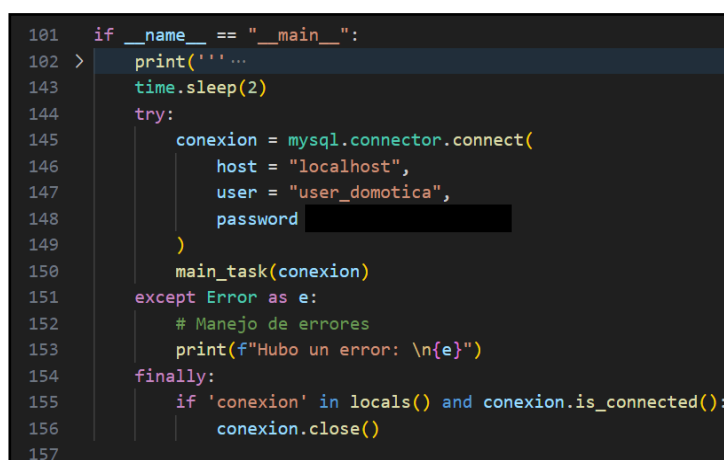
Este archivo se va a ejecutar automáticamente cada 5 minutos en todos los POS que cuenten con sistema operativo Ubuntu debido a que se encuentra versionado a través del repositorio de Git llamado “servicios”. Su ejecución se realiza a través del crontab:

```
* /5 * * * * /usr/bin/python3 /servicios/Domotica/codigo_principal.py
```

B. Explicación del código

1) Inicio

El inicio del código tiene varias partes importantes según la siguiente imagen:



```

101 if __name__ == "__main__":
102 >     print('...')
103     time.sleep(2)
104     try:
105         conexion = mysql.connector.connect(
106             host = "localhost",
107             user = "user_domotica",
108             password = "password"
109         )
110         main_task(conexion)
111     except Error as e:
112         # Manejo de errores
113         print(f"Hubo un error: \n{e}")
114     finally:
115         if 'conexion' in locals() and conexion.is_connected():
116             conexion.close()
117 
```

Imagen 15. Inicio código en Python.

1. **try: ... except: ... finally:** Este es un bloque de manejo de errores:
 - a. **try:** intenta ejecutar el código dentro de este bloque. En este caso, intenta realizar una conexión a la base de datos MySQL.
 - b. **except Error as e:** si ocurre un error durante la conexión, este bloque captura el error y lo imprime en la consola. La variable *e* contiene el detalle del error.
 - c. **finally:** este bloque se ejecuta siempre, independientemente de si hubo un error o no. El bloque verifica si la variable *conexion* fue creada y está conectada; si es así, cierra la conexión a la base de datos para liberar los recursos.
2. **Conexión a la base de datos:** Se especifica el servidor (en este caso localhost, lo que significa que la base de datos está en la misma máquina), el usuario (*user_domotica*) y la contraseña.
3. **main_task(conexion):** Aquí se llama a una función llamada *main_task*, a la cual se le pasa la variable *conexion* como argumento.

2) Función main_task

Realiza varias operaciones utilizando una conexión a una base de datos y luego ejecuta comandos basados en los datos recuperados.

```
def main_task(conexion):
    config = consulta_configuracion(conexion)
    if config == 'Automatico':
        try:
            consulta_eventos(conexion)
            estados = consulta_estados(conexion)
            for _ in range(5):
                print("\n##### ENVIANDO ESTADOS #####\n")
                for estado in estados:
                    ip = str(estado[0])
                    pin = estado[1]
                    state = str(estado[2])
                    if pin in [0,1,2,3,4] and state in ['ON', 'OFF']:
                        if state == 'ON':
                            state = 1
                        else:
                            state = 0
                        send_command(ip, pin, state)
                    else:
                        print("Entrada no válida. Por favor, ingresa a la base de datos un número de pin entre (0-4) y estado (ON o OFF).")
                print("\nLOS ESTADOS SE HAN CAMBIADO CORRECTAMENTE\n")
            except KeyboardInterrupt:
                print("Saliendo.")
        else:
            print(f'\nEn estos momentos el sistema se cuenta en modo {config}\n')
```

Imagen 16. Función main_task

1. Consulta la configuración: ejecuta la función *consulta_configuración* y guarda su *return* en la variable *config*. En caso de que el sistema está en modo "Automático", realiza las operaciones, sino, envía un mensaje y no realiza ninguna de las demás funciones.
2. *try: ... except:* el bloque *try* intenta ejecutar las operaciones principales de la función, mientras que el *except KeyboardInterrupt:* captura una interrupción manual del programa (por ejemplo, si el usuario presiona Ctrl + C para detener la ejecución). Si eso sucede, imprime "Saliendo." y termina el programa de manera limpia.
3. *consulta_eventos(conexion)*: esta función se utiliza para consultar eventos desde la base de datos y recibe la variable *conexion* como argumento.
4. *estados = consulta_estados(conexion)*: se llama a otra función, *consulta_estados*, que recibe la conexión a la base de datos como argumento y retorna una lista de "estados" almacenados. Cada estado contiene 3 datos.
5. *Bucle for estado in estados*: este bucle recorre cada uno de los "estados" obtenidos de la base de datos. Para cada estado:
 - a. *ip* toma el valor de la dirección IP del dispositivo.
 - b. *pin* toma el número del pin (un entero).
 - c. *state* toma el valor de estado del pin (si está "ON" o "OFF").
6. *Validación del pin y state*: se verifica si el pin es un número válido entre 0 y 4, y si el state es "ON" o "OFF". Si cualquiera de estas condiciones no se cumple, se imprime un mensaje de error.
7. *Conversión del estado*: se convierte el estado de "ON" o "OFF" a un valor numérico: 1 para "ON" y 0 para "OFF". Esto es necesario porque la función que envía comandos a los dispositivos espera valores numéricos para el estado.
8. *send_command(ip, pin, state)*: llama a la función *send_command* que recibe como parámetros la ip, el pin y el estado hallados anteriormente.

3) Función consulta_eventos

Esta función como objetivo actualizar los estados de dispositivos en función de ciertos eventos programados en la base de datos.

```

6 def consulta_eventos(conexion):
7     print("\n##### CONSULTANDO EVENTOS Y ACTUALIZANDO ESTADOS #####\n")
8     update = '''
9     UPDATE domotica.domotica_onoff AS d
10    JOIN (
11        SELECT
12            e.domotica_onoff_fk,
13            MAX(CASE
14                WHEN @hora_actual >= e.hora_encendido AND @hora_actual < e.hora_apagado THEN 'ON'
15                ELSE 'OFF'
16            END) AS estado
17        FROM domotica.domotica_eventos e
18        GROUP BY e.domotica_onoff_fk
19    ) AS sub1
20    ON d.dk = sub1.domotica_onoff_fk
21    SET d.estado = sub1.estado;
22    '''
23    try:
24        if conexion.is_connected():
25            cursor = conexion.cursor()
26            cursor.execute("SET @hora_actual = curtime();")
27            conexion.commit()
28            cursor.execute(update)
29            conexion.commit()
30            print("Los estados han sido actualizados correctamente.")
31
32    except Error as e:
33        # Manejo de errores
34        print(f"Hubo un error al consultar los eventos: \n{e}")
35    finally:
36        # Cerrar el cursor y la conexión
37        if 'cursor' in locals() and cursor:
38            cursor.close()

```

Imagen 17. Función consulta_eventos

1. *Consulta SQL de actualización (UPDATE):* define una consulta SQL que tiene como objetivo actualizar los estados de ciertos dispositivos en función de eventos registrados en la base de datos.
 - a. *Tabla principal (domotica.domotica_onoff):* Se actualiza la columna estado de la tabla domotica_onoff.
 - b. *Subconsulta sub1:* La subconsulta selecciona las entradas de la tabla domotica_eventos, que almacena eventos con horas de encendido (hora_encendido) y apagado (hora_apagado) para los dispositivos.
 - c. *Cálculo del estado (ON/OFF) basado en la hora actual:*
 - i. La subconsulta usa un CASE para determinar si, en función de la hora actual (@hora_actual), el dispositivo debe estar encendido ("ON") o apagado ("OFF"). Si la hora actual está entre hora_encendido y hora_apagado, se asigna el estado "ON"; en cualquier otro caso, se asigna "OFF".
 - ii. El uso de MAX(CASE ...) garantiza que se elija el valor más reciente en caso de que haya múltiples eventos para un dispositivo.
2. *Ejecución del código SQL:*
 - a. *cursor = conexion.cursor():* Se crea un cursor que permite interactuar con la base de datos.
 - b. *cursor.execute("SET @hora_actual = curtime();"):* Se asigna la hora actual del sistema a la variable @hora_actual para su uso en la consulta. curtime() es una función SQL que retorna la hora actual en formato de tiempo (HH:MM:SS).
 - c. *cursor.execute(update):* Ejecuta la consulta SQL definida previamente, la cual actualiza los estados de los dispositivos en la tabla domotica_onoff
 - d. *conexion.commit():* Confirma los cambios en la base de datos, garantizando que los estados se actualicen.
3. *Cierre de recursos:* El bloque finally asegura que el cursor se cierre correctamente una vez que se haya terminado de usar:

4) Función consulta_estados

La función tiene como objetivo obtener y devolver los estados de ciertos dispositivos domóticos almacenados en la base de datos.

```

40 def consulta_estados(conexion): #utilizar una consulta donde se traiga cada dato y lo ponga en una lista
41     print("\n##### CONSULTANDO ESTADOS #####\n")
42     #Buscar la consulta de productos
43     consulta = '''
44     select
45         dd.dir_ip,
46         donoff.pin_id,
47         donoff.estado
48     from domotica.domotica_onoff as donoff
49     inner join domotica.domotica_dispositivos as dd on donoff.domotica_dispositivos_fk = dd.dk;
50     '''
51     #Realizar la conexión con la base de datos
52     try:
53         cursor = conexion.cursor()
54         cursor.execute(consulta)
55         estados=cursor.fetchall()
56         return estados
57     except Error as e:
58         print(f"Hubo un error al actualizar los estados: \n{e}")
59     finally:
60         if 'cursor' in locals() and cursor:
61             cursor.close()

```

Imagen 18. Función consulta estados.

1. *Consulta SQL:*

- a. Relaciona los campos de la dirección IP del dispositivo, el número de pin que se está controlando y el estado actual del pin, que indica si está "ON" o "OFF".
- b. El uso de INNER JOIN entre las tablas domotica_onoff y domotica_dispositivos se basa en la clave foránea domotica_dispositivos_fk, que enlaza el dispositivo con sus pines correspondientes. La clave primaria en la tabla de dispositivos es dd.dk.

2. *Ejecución de la consulta:*

- a. `cursor = conexion.cursor()`: Se crea un cursor para interactuar con la base de datos.
- b. `cursor.execute(consulta)`: Ejecuta la consulta SQL que se ha definido previamente.
- c. `estados = cursor.fetchall()`: Recupera todos los resultados de la consulta y los almacena en la variable estados. Este método devuelve una lista de tuplas, donde cada tupla contiene los valores de cada fila obtenida.

3. *Cierre del cursor*: El bloque finally garantiza que, pase lo que pase (ya sea que la consulta sea exitosa o ocurra un error), el cursor se cierre correctamente al final de la función.

5) *Función send_command*

Esta función recibe como parámetros la IP en formato string, el pin en formato entero y el estado del pin en formato string; a su vez, envía comandos HTTP a un dispositivo específico en una red local para controlar los pines requeridos.

```

63 def send_command(ip:str, pin:int, state:str):
64
65     # Construir la URL con los parámetros proporcionados por el usuario
66     url = f'http://{ip}/control?pin={pin}&state={state}'
67
68     try:
69         # Enviar la solicitud GET
70         response = requests.get(url)
71         # Verificar el código de estado HTTP y mostrar el resultado
72         if response.status_code == 200:
73             print(f"Enviando comando - IP: {ip}, Pin: {pin}, Estado: {state}")
74         else:
75             print("Error enviando comando")
76     except requests.RequestException:
77         # Manejar cualquier error de solicitud sin mostrar detalles
78         print("Error en enviar la solicitud al dispositivo.")

```

Imagen 19. Función send_command

1. *Construcción URL*: Esta línea construye una URL que sigue el formato de una solicitud GET al dispositivo que tiene la IP proporcionada. La URL generada tiene la siguiente estructura:

http://[IP_del_dispositivo]/control?pin=[numero_pin]&state=[estado]

2. *Envío de la solicitud*: Se usa la librería *requests* de Python para enviar una solicitud HTTP GET a la URL que se acaba de construir. Es donde el programa intenta comunicar el comando al dispositivo. Si el dispositivo responde correctamente, este comando debería activar o desactivar el pin indicado.
3. *Verificación de la respuesta HTTP*: Se verifica el código de estado HTTP devuelto por el dispositivo en `response.status_code`
4. *Manejo de excepciones*: este bloque `except` captura cualquier excepción que pueda surgir al intentar realizar la solicitud HTTP, como por ejemplo si el dispositivo no está disponible, si hay problemas de red, o si la dirección IP es incorrecta. La clase `RequestException` captura una variedad de posibles errores, como fallos de conexión, tiempos de espera, y errores de DNS.

6) Función consulta_configuracion:

Es una función que toma como argumento conexión, el cual representa la conexión activa a la base de datos, y consulta a la base de datos para obtener el modo de operación del sistema.

```

def consulta_configuracion(conexion):
    print("\n##### CONSULTANDO LA CONFIGURACION DEL SISTEMA #####\n")
    consulta = '''
        select valor from domotica_configuracion where alias = 'Modo';
        ...
    '''
    try:
        cursor = conexion.cursor()
        cursor.execute(consulta)
        config = cursor.fetchone()
        print ("Configuracion consultada correctamente\n")
        if config:
            valor = str(config[0]).strip()
            return valor
        else:
            print("No se encontró configuración del sistema")
    except Error as e:
        print(f"Hubo un error al consultar la configuracion del sistema: \n{e}")
    finally:
        if 'cursor' in locals() and cursor:
            cursor.close()

```

Imagen 20. Función consulta_configuracion.

1. Con el objeto cursor, se ejecuta la consulta SQL definida previamente usando el método `execute`.
2. Se usa el método `fetchone` para obtener la primera fila del resultado de la consulta. Este resultado se almacena en la variable `config`.
3. Si la variable `config` contiene datos, se toma el primer elemento de `config` (el valor consultado), se convierte a una cadena `str` y se eliminan espacios en blanco innecesarios usando `strip()`. Este valor se retorna como resultado de la función.
4. Si ocurre un error durante la consulta, el bloque `except` captura la excepción y se imprime un mensaje con los detalles del error.
5. En el bloque `finally`, se asegura que el cursor sea cerrado para liberar recursos. Antes de cerrarlo, se verifica que el cursor exista y esté activo.