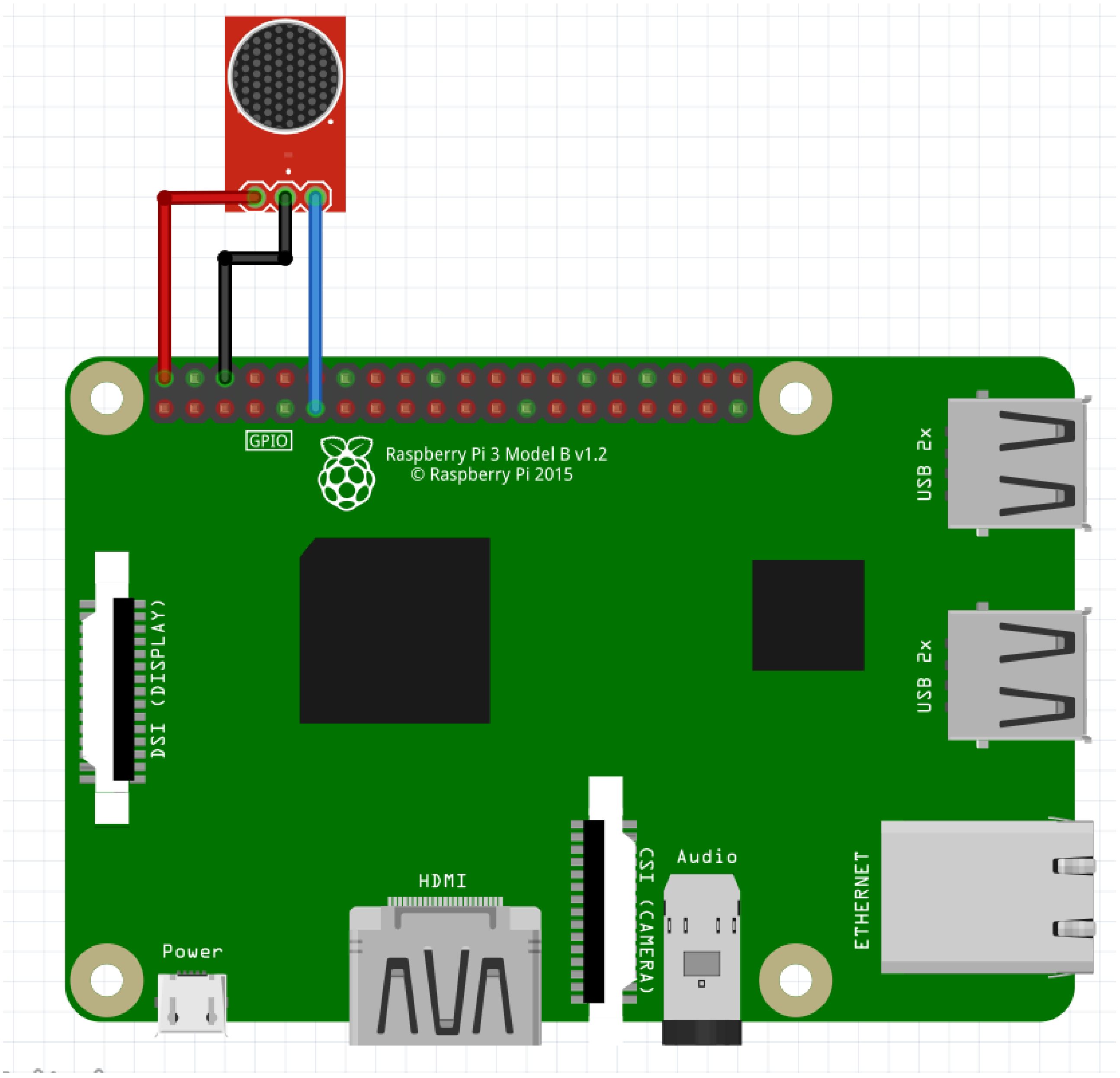
Phase 3: Report submission:

Noise pollution monitoring:

Introduction:

As urban areas continue to grow and expand, so does the concern over noise pollution. To address this challenge, the development of an IoT-enabled Noise Pollution Monitoring system is crucial. Such a system leverages IoT sensors placed in public areas to measure noise levels in real-time and collect valuable data.This project aims to provide an illustrative example of how such a system can be implemented. A synthetic dataset has been created to simulate noise level data, and a Python script has been developed to demonstrate how an IoT sensor can send this data to a noise pollution information platform. The data collected from these sensors can be used to monitor noise levels, identify trends, and make informed decisions for noise pollution control and management.

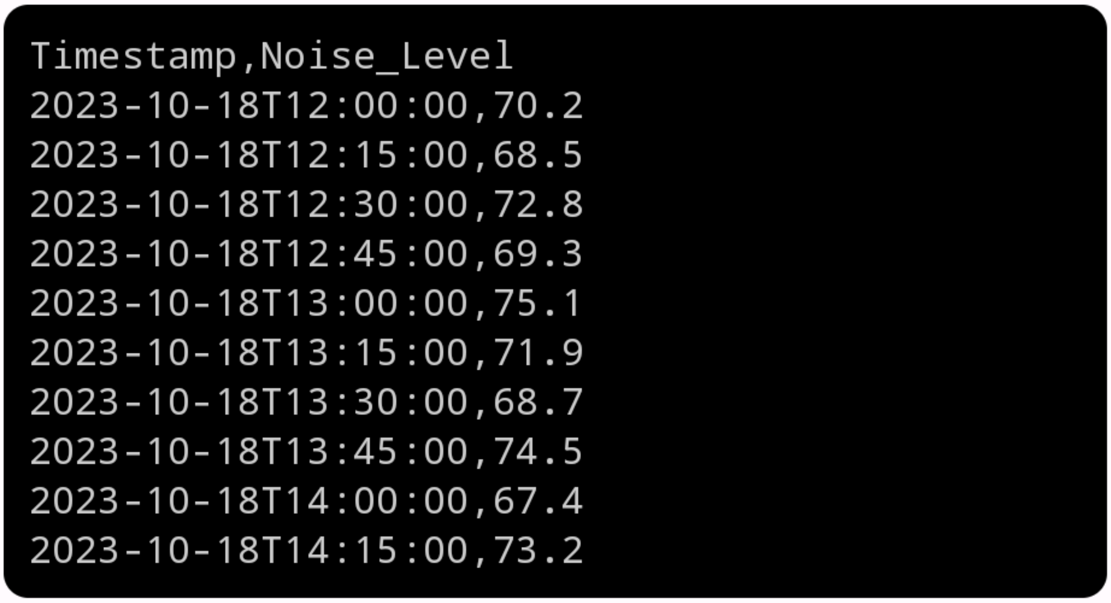
Circuit diagram:



Circuit diagram explanation :

In the context of an IoT-enabled Noise Pollution Monitoring system, the use of a sound sensor, specifically a sound level sensor or microphone, is a fundamental component. This sensor is employed in conjunction with a Raspberry Pi to continuously monitor and assess fluctuations in the acoustic environment.The sound sensor is a sensitive component designed to detect and convert sound waves into electrical signals. It captures sound data from its surroundings and transmits it to the Raspberry Pi for analysis and processing. The Raspberry Pi serves as the brain of the system, providing the necessary computational power to receive, interpret, and store the incoming noise data.

Dataset:

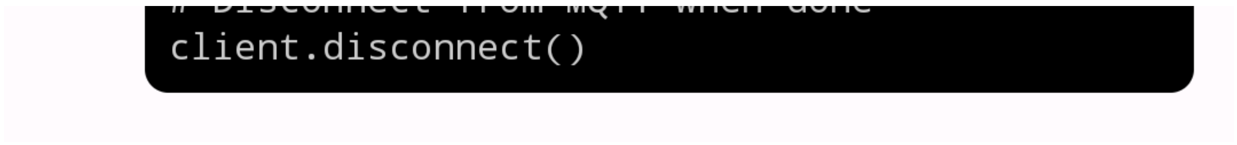


```
Timestamp,Noise_Level
2023-10-18T12:00:00,70.2
2023-10-18T12:15:00,68.5
2023-10-18T12:30:00,72.8
2023-10-18T12:45:00,69.3
2023-10-18T13:00:00,75.1
2023-10-18T13:15:00,71.9
2023-10-18T13:30:00,68.7
2023-10-18T13:45:00,74.5
2023-10-18T14:00:00,67.4
2023-10-18T14:15:00,73.2
```

You can save this data to a CSV file and use it as your dataset. To create a larger dataset, you can use the Python code to generate more timestamps and random noise levels as needed.

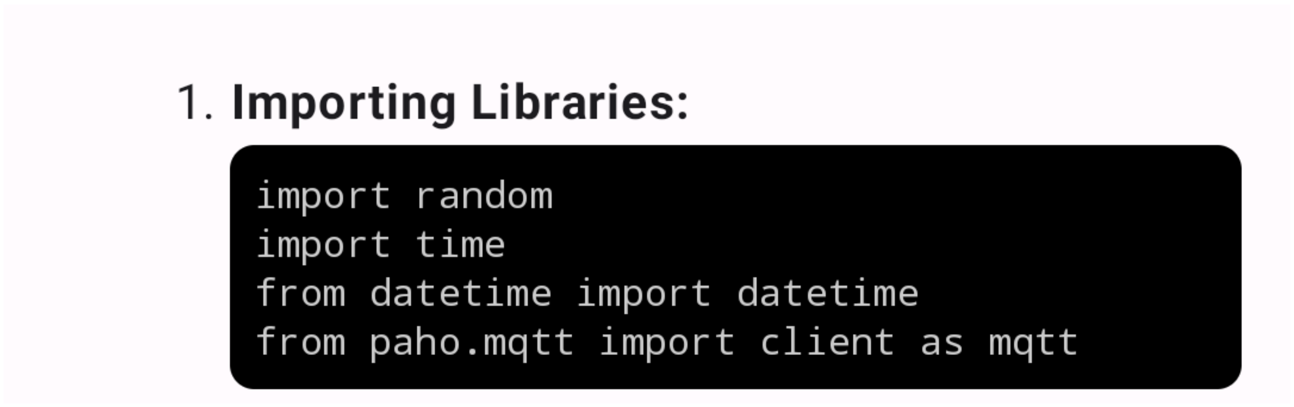
Code:

```
import random
import time
from datetime import datetime
from paho.mqtt import client as mqtt

# Simulated IoT Sensor
def simulate_noise_sensor():
    return random.uniform(40, 100)  #
Simulating noise level data

# MQTT Broker Configuration (you may need to
modify this for your chosen platform)
mqtt_broker = "broker.example.com"
mqtt_port = 1883
mqtt_topic = "noise_data_topic"
client_id = "noise_sensor_client"

# Initialize MQTT Client
client = mqtt.Client(client_id)

# Connect to the MQTT Broker
client.connect(mqtt_broker, mqtt_port,
keepalive=60)

# Simulate and Send Noise Data
while True:
    noise_level = simulate_noise_sensor()
    timestamp = datetime.now().isoformat()
    payload = f"{{'timestamp': '{timestamp}',
'noise_level': {noise_level}}}"

    # Publish data to the IoT platform
    client.publish(mqtt_topic, payload)

    print(f"Published: {payload}")

    time.sleep(10)  # Simulate sending data
every 10 seconds

# Disconnect from MQTT when done
```



```
# Disconnect from MQTT when done
client.disconnect()
```

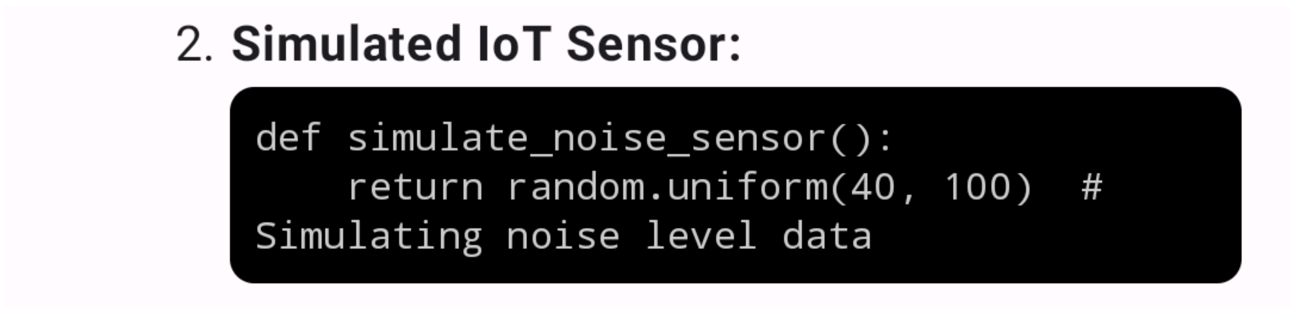
Code explanation:



1. **Importing Libraries:**

```
import random
import time
from datetime import datetime
from paho.mqtt import client as mqtt
```

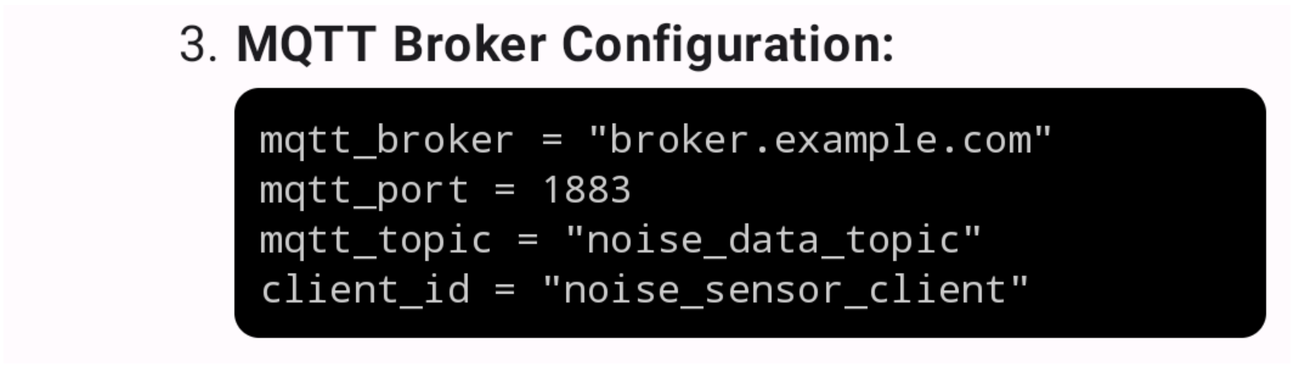
- We import necessary libraries for generating random numbers (random), working with time (time), handling timestamps (datetime), and connecting to an MQTT broker (paho.mqtt).



2. **Simulated IoT Sensor:**

```
def simulate_noise_sensor():
    return random.uniform(40, 100)  #
Simulating noise level data
```

- This function simulate_noise_sensor generates a random noise level data between 40 and 100 (you can adjust this range to simulate realistic noise levels).



3. **MQTT Broker Configuration:**

```
mqtt_broker = "broker.example.com"
mqtt_port = 1883
mqtt_topic = "noise_data_topic"
client_id = "noise_sensor_client"
```

- Here, we specify the MQTT broker's address, port, the topic to which we will publish the data, and a client ID for the IoT sensor.

4. **Initialize MQTT Client:**

```
client = mqtt.Client(client_id)
```

- We create an MQTT client with the specified client ID.

5. **Connect to MQTT Broker:**

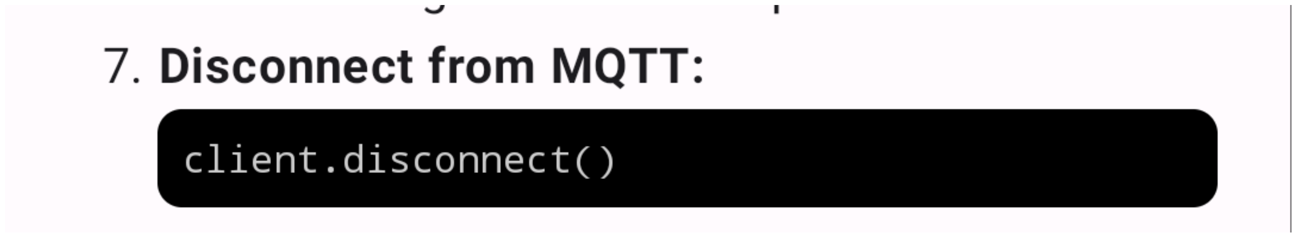
```
client.connect(mqtt_broker, mqtt_port,
keepalive=60)
```

- The script establishes a connection to the MQTT broker using the provided address and port.

6. **Simulate and Send Noise Data:**

```
while True:
    noise_level = simulate_noise_sensor()
    timestamp = datetime.now().isoformat()
    payload = f"{{'timestamp':
'{timestamp}', 'noise_level':
{noise_level}}}"

    client.publish(mqtt_topic, payload)

    print(f"Published: {payload}")

    time.sleep(10)
```

- In this loop, the script simulates noise data:It generates a noise level using the simulate_noise_sensor function.It creates a timestamp using the current time and formats it as an ISO string.It assembles a JSON payload with the timestamp and noise level.The script publishes this payload to the MQTT topic.It prints the sent data for reference.It then waits for 10 seconds before sending the next data point.


7. **Disconnect from MQTT:**

```
client.disconnect()
```

- When the script is done (which might not happen in this infinite loop), it disconnects from the MQTT broker.

Conclusion:

In conclusion, the integration of a sound sensor with a Raspberry Pi creates a powerful tool for real-time noise pollution monitoring, offering insights into the acoustic environment, and contributing to better-informed decisions regarding noise management and urban planning.