PHASE:3 NOISE POLLUTION MONITORING

DEVELOPMENT PART:1

* Start building the IOT enabled noise pollution monitoring system

Creating the entire codebase for an IoT-enabled noise pollution monitoring system is a complex task that goes beyond the scope of a single response. However, I can provide you with a simplified example of code that you can use as a starting point. This example will demonstrate how to read data from a simple microphone sensor connected to a Raspberry Pi and send it to a server. Please note that this is a basic implementation, and a real-world system would be more complex.

Here’s a simplified Python code snippet to get you started:

Import os

Import sounddevice as sd

Import numpy as np

Import requests

# Function to record audio and send data to the server

Def monitor\_noise():

While True:

# Record audio for a few seconds

Duration = 5 # Adjust as needed

Fs = 44100 # Sample rate

Audio\_data, \_ = sd.rec(int(duration \* fs), samplerate=fs, channels=1, dtype=’int16’)

Sd.wait()

# Calculate noise level (e.g., RMS)

Noise\_level = np.sqrt(np.mean(np.square(audio\_data))

# Send data to the server

Data = {

“sensor\_id”: sensor\_id,

“noise\_level”: noise\_level

}

Response = requests.post(server\_url, json=data)

If response.status\_code == 200:

Print(f”Data sent to server: {data}”)

Else:

Print(f”Failed to send data to server. Error: {response.status\_code}”)

If \_\_name\_\_ == “\_\_main\_\_”:

Monitor\_noise()

* SENSOR DEPLOYMENT:

Sensor deployment in an IoT-enabled noise pollution monitoring system typically involves configuring and installing sensors in the desired locations. Here’s a simplified Python code snippet that simulates sensor deployment. In a real-world scenario, you would need to physically install the sensors and configure them accordingly.

Class NoiseSensor:

Def \_\_init\_\_(self, sensor\_id, location):

Self.sensor\_id = sensor\_id

Self.location = location

Def read\_noise\_level(self):

# Simulate reading noise levels from the sensor

# In a real deployment, this would involve interacting with the sensor hardware

Import random

Noise\_level = random.uniform(40, 100) # Simulated noise level in dB(A)

Return noise\_level

Def deploy\_sensors():

# Define sensor locations and IDs

Sensor\_locations = {

“sensor1”: “Location A”,

“sensor2”: “Location B”,

}

Sensors = []

# Create sensor instances

For sensor\_id, location in sensor\_locations.items():

Sensor = NoiseSensor(sensor\_id, location)

Sensors.append(sensor)

# Read and print noise levels from each sensor

For sensor in sensors:

Noise\_level = sensor.read\_noise\_level()

Print(f”Sensor ID: {sensor.sensor\_id}, Location: {sensor.location}, Noise Level: {noise\_level} dB(A)”)

If \_\_name\_\_ == “\_\_main\_\_”:

Deploy\_sensors()

* DATA ACQUISITION:

Data acquisition in an IoT-enabled noise pollution monitoring system involves collecting data from sensors. In a real-world implementation, you would interface with specific sensor hardware, but I can provide you with a simple example of data acquisition code that reads simulated noise levels from sensors.

Import time

Import random

Class NoiseSensor:

Def \_\_init\_\_(self, sensor\_id, location):

Self.sensor\_id = sensor\_id

Self.location = location

Def read\_noise\_level(self):

# Simulate reading noise levels from the sensor

Noise\_level = random.uniform(40, 100) # Simulated noise level in dB(A)

Return noise\_level

Def data\_acquisition():

# Define sensor locations and IDs

Sensor\_locations = {

“sensor1”: “Location A”,

“sensor2”: “Location B”,

“sensor3”: “Location C”,

}

Sensors = []

# Create sensor instances

For sensor\_id, location in sensor\_locations.items():

Sensor = NoiseSensor(sensor\_id, location)

Sensors.append(sensor)

# Simulate continuous data acquisition

While True:

For sensor in sensors:

Noise\_level = sensor.read\_noise\_level()

Timestamp = time.strftime(“%Y-%m-%d %H:%M:%S”)

Print(f”Timestamp: {timestamp}, Sensor ID: {sensor.sensor\_id}, Location: {sensor.location}, Noise Level: {noise\_level} dB(A)”)

# Adjust the acquisition interval

Time.sleep(5) # Adjust as needed

If \_\_name\_\_ == “\_\_main\_\_”:

Data\_acquisition()

* ALERTING SYSTEM:

Certainly, here’s a simplified Python code snippet for an alerting system that sends an alert message if a noise level threshold is exceeded

Class NoiseSensor:

Def \_\_init\_\_(self, sensor\_id, location, threshold):

Self.sensor\_id = sensor\_id

Self.location = location

Self.threshold = threshold

Def read\_noise\_level(self):

# Simulate reading noise levels from the sensor

Import random

Noise\_level = random.uniform(40, 100) # Simulated noise level in dB(A)

Return noise\_level

Def send\_alert(sensor, noise\_level):

If noise\_level > sensor.threshold:

Print(f”ALERT: Sensor {sensor.sensor\_id} at {sensor.location} detected noise level {noise\_level} dB(A) exceeding the threshold of {sensor.threshold} dB(A)”)

Def alerting\_system():

Sensor1 = NoiseSensor(“sensor1”, “Location A”, 70)

Sensor2 = NoiseSensor(“sensor2”, “Location B”, 65)

Sensor3 = NoiseSensor(“sensor3”, “Location C”, 75)

While True:

Noise\_level1 = sensor1.read\_noise\_level()

Noise\_level2 = sensor2.read\_noise\_level()

Noise\_level3 = sensor3.read\_noise\_level()

Send\_alert(sensor1, noise\_level1)

Send\_alert(sensor2, noise\_level2)

Send\_alert(sensor3, noise\_level3)

If \_\_name\_\_ == “\_\_main\_\_”:

Alerting\_system()

* User interface:

Creating a user interface for your noise pollution monitoring system typically involves using web-based technologies. Here’s a simplified example of how to create a basic web-based user interface using HTML and Python’s Flask framework for a local implementation

from flask import Flask, render\_template

Import random

App = Flask(\_\_name\_\_)

# Simulated sensor data

Def get\_sensor\_data():

Sensor\_data = {

“sensor1”: random.uniform(40, 100),

“sensor2”: random.uniform(40, 100),

“sensor3”: random.uniform(40, 100),

}

Return sensor\_data

Def index():

Sensor\_data = get\_sensor\_data()

Return render\_template(‘index.html’, sensor\_data=sensor\_data)

If \_\_name\_\_ == ‘\_\_main\_\_’:

App.run(debug=True)