**SMART AC RECOMMENDATION SYSTEM**

**INTRODUCTION :**

Imagine having an intelligent assistant that not only keeps you comfortable but also helps you save energy. Our AIoT project brings this vision to life by creating a seamless connection between the outside world, your home, and smart decision-making.

We gather real-time weather data like temperature and humidity from the world outside, and at the same time, we keep track of the indoor climate using specialized sensors. We're even attentive to whether your AC is running or not, because every little detail matters.

All of this valuable information gets stored in a digital vault called a time series database. It's like a giant digital journal that captures how things change over time. But the real magic happens when we use advanced AI techniques to analyze all this data. It's like teaching our system to understand patterns and predict the future, but for home comfort!

When you have questions or need advice, our friendly AI assistant, powered by ChatGPT, is here to help. Wondering when to turn on the AC to avoid heat or humidity? Just ask! Our AI assistant dives into the treasure trove of data, looks at how the weather has been behaving, how comfy your home is right now, and even what your AC has been up to. Then, it cleverly suggests the best times to switch on the AC for maximum comfort and energy savings.But that's not all – we make sure you receive the advice in a way that suits you. Our AI can talk! You'll get to hear its recommendations spoken aloud, just like having a conversation. And if you prefer, you'll also receive the advice through your Telegram app.

What's truly exciting is that this system learns and evolves. The more people use it, the smarter it gets at predicting the best AC times. So, it's not just about making you feel cozy – it's about making homes smarter, saving energy, and contributing to a happier planet. Welcome to the future of intelligent home comfort!

**HARDWARE USED:**

1. WIZnet-W5300 TOE SHIELD + STM32-F429ZI board
2. DHT11
3. Jumperwires
4. Bread Board
5. Microphone
6. Resistor

**SOFTWARE USED:**

* MQTT SERVICE
* PYCHARM
* ARDUINO IDE

**PROGRAMMING LANGUAGES:**

* C++
* PYTHON

**APPS AND ONLINE SERVICES USED:**

* MQTT BROKER
* OPENAI
* INFLUXDB
* TELEGRAM API

****

**WORKFLOW:**

1. User Voice Command:

- The user interacts with the system by giving a voice command through a device, such as a smartphone or a smart speaker. For instance, they might say, "When should I turn on the AC today?"

2. Speech-to-Text Conversion:

- The voice command is captured and converted into text using a speech-to-text technology. This transforms the spoken words into a format that can be processed by the AI system.

3. Query Sent to Data Retrieval Module:

- The converted text query is passed to a data retrieval module, which contains Python code and queries specifically designed to fetch historical time series data from the InfluxDB database. This data includes details about external weather conditions, indoor climate measurements, AC status, and other relevant variables recorded over time.

4. Data Sent to OpenAI Model:

- Once the relevant data is fetched from the InfluxDB database, it's sent to the OpenAI model as input. The data provides context for the user's query and allows the AI model to understand and analyze the user's question accurately.

5. Data Analysis by OpenAI:

- Using the received historical time series data, the OpenAI model processes the information to identify patterns, correlations, and trends. It considers factors like past AC usage, weather trends, and comfort levels to formulate a suitable response.

6. Text-to-Speech Conversion:

- After analyzing the data, the OpenAI model generates a response text suggesting the optimal time to turn on the AC. This response text is converted into speech using a text-to-speech synthesis tool.

7. Voice Response to User:

- The synthesized voice response is played back to the user through a speaker or audio output device. The user hears the AI's recommendation spoken aloud, providing a natural and convenient interaction.

8. Telegram Bot Notification:

- Simultaneously, the response generated by the OpenAI model is sent to the Telegram bot. The bot then forwards the recommendation as a text message to the user's Telegram account.

**Flowchart:**

**SETTING UP USING PYTHON:**

**Setting up with MQTT:**

import paho.mqtt.client as mqtt

paho.mqtt.client: This library provides a client implementation for MQTT (Message Queuing Telemetry Transport), a lightweight messaging protocol widely used in the Internet of Things (IoT) domain for communication between devices. It allows you to connect to an MQTT broker and publish messages to topics or subscribe to topics to receive messages. It's commonly used for real-time data transmission in IoT applications.

Installation: To install the paho-mqtt library, you can use pip, the Python package manager. Open your terminal or command prompt and run the following command:

pip install paho-mqtt

**Setting up with OPENAI:**

import openai

openai: This library provides a Python interface for the OpenAI API, allowing you to interact with various natural language processing models and services provided by OpenAI. With this library, you can access powerful language models like GPT-3 to perform tasks such as text generation, language translation, question-answering, and more.

Installation: To install the openai library, you can use pip as well. Run the following command in your terminal or command prompt:

pip install openai

**Setting Up with Recogniser:**

import speech\_recognition as sr

This library provides an easy-to-use interface to work with various speech recognition APIs. It allows you to convert spoken language into text and supports multiple speech recognition engines, such as Google Web Speech API, Microsoft Bing Voice Recognition, and more.

Installation: To install the speech\_recognition library, you can use pip:

pip install SpeechRecognition

**Setting up with InfluxDB:**

import influxdb\_client  
from influxdb\_client import InfluxDBClient, Point, WritePrecision  
from influxdb\_client.client.write\_api import SYNCHRONOUS

1. influxdb\_client:
   * This library allows you to interact with InfluxDB, a time series database, using Python. It provides functionalities to read and write data to the database, perform queries, and manage database operations.
2. InfluxDBClient, Point, WritePrecision:
   * InfluxDBClient is a class that provides the necessary methods to establish a connection to an InfluxDB server. It allows you to configure connection settings such as the server's URL and authentication credentials.
   * Point represents a data point in InfluxDB. It lets you define measurements, tags, fields, and a timestamp for the data you want to write.
   * WritePrecision is an enumeration that specifies the precision of the timestamp when writing data points to InfluxDB.

**Setting up with Datetime and timedelta:**

from datetime import timedelta, datetime

* **datetime** is a module that supplies classes for manipulating dates and times. It's used to work with timestamps and time-related calculations.
* **timedelta** is a class within the **datetime** module that represents the difference between two dates or times. It's useful for adding or subtracting time intervals.

**Setting up with TTS:**

import pyttsx3

This library is a text-to-speech (TTS) engine that allows you to convert text into speech. It's useful for applications where you want your computer or device to speak out information to the user.

Installation: To install the pyttsx3 library, you can use pip:

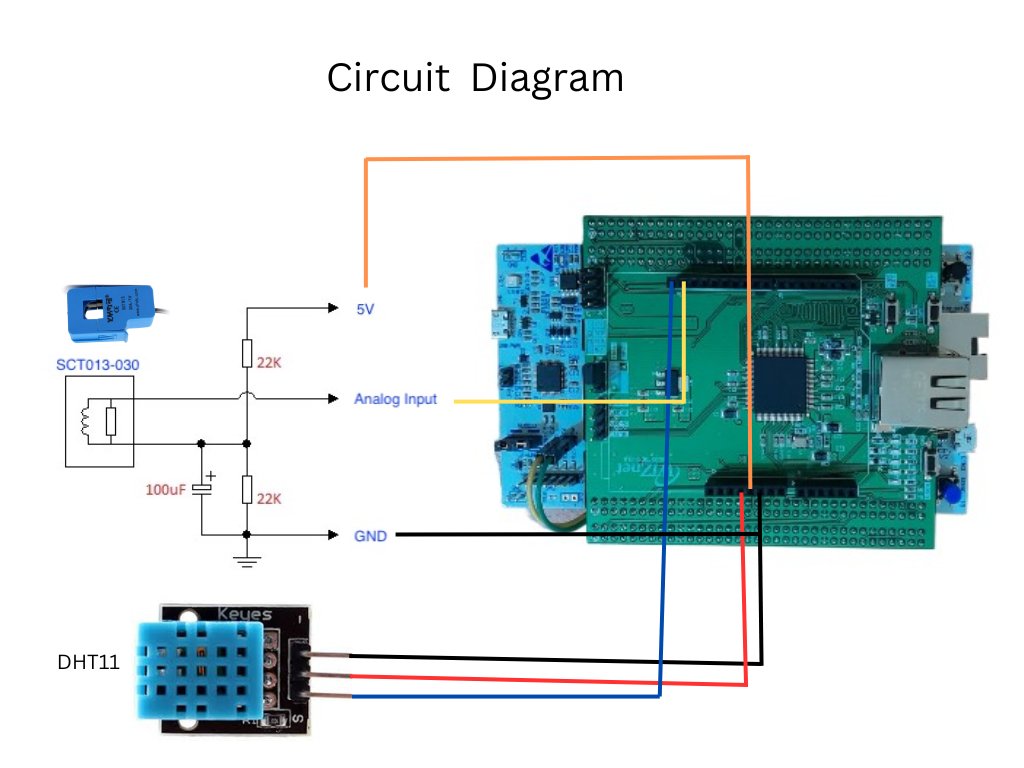
pip install pyttsx3

**Setting up with Telegram:**

import telebot

Telebot is a Python library that interacts with the Telegram Bot API. It simplifies the process of creating and managing Telegram bots. You can use it to send and receive messages, multimedia, and other content through Telegram bot interactions.

CIRCUIT DIAGRAM:



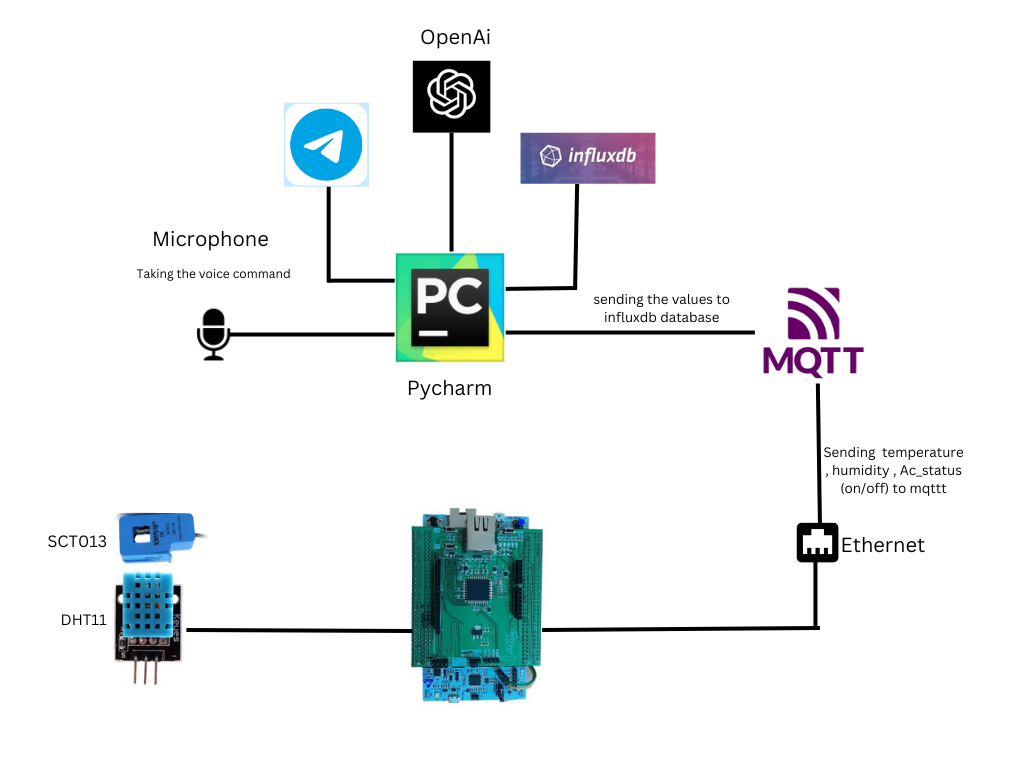
The DHT11 (Humidity – temperature Sensor) has 3 pins GND , VCC , Signal

* VCC is connected with 3.3v
* GND is connected with Gnd
* Signal is connected with D15

The SCTO13 that whole connection as we can see has 3 pins GND , Output , VCC

* VCC is connected with 5v
* GND is connected with same Gnd
* Output pin is connected with D14

CONNECTION DIAGRAM :



SAMPLE UTTERANCES:

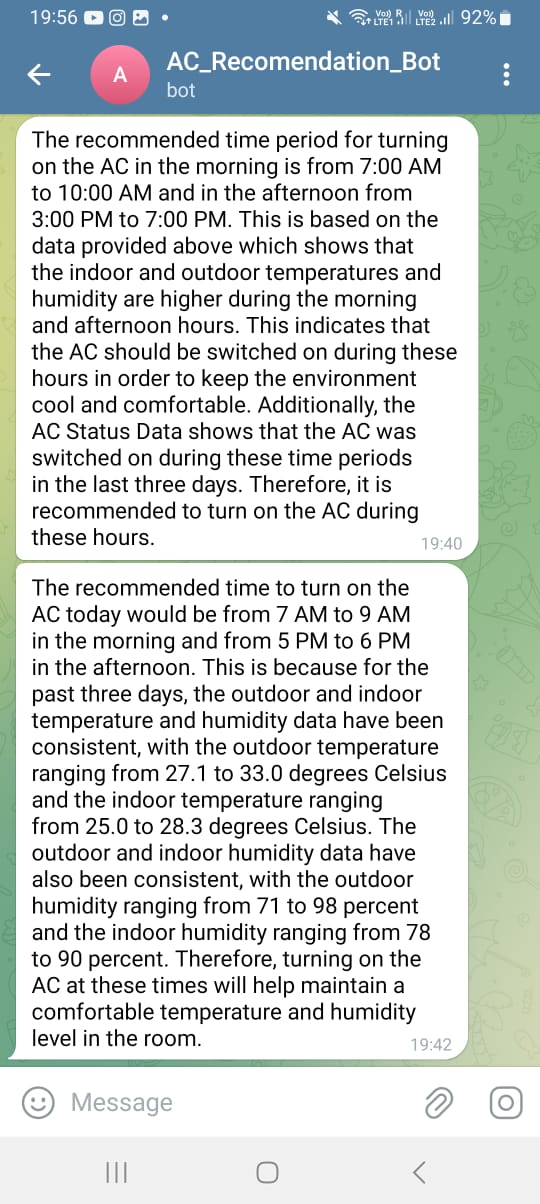
I have tested the below given utterances for opening and closing and it was working fine. More phrases can be possible:

Recommendation:

* System What is the right time to turn on the AC today
* System What is the best time to turn on the AC today
* System What is the recommended time to turn on the Ac
* System Suggest a Suitable table to turn on the AC
* System Recommend best time to turn on the AC today

Output:

Its giving voice responses and also giving responses in telegram



Output :

Video :

**Code Explanation:**

**Python source code :**

**Sending values to influxdb :**

**Code:**

import requests  
import time  
import influxdb\_client  
import paho.mqtt.client as mqtt  
from influxdb\_client import InfluxDBClient, Point, WritePrecision  
from influxdb\_client.client.write\_api import SYNCHRONOUS  
  
  
class InfluxDB:  
 def \_\_init\_\_(self, bucket="Weather\_Test\_Data"):  
 self.token = "4S-CyOmTPUhSgUiUSf0A8Fn31keTgErGXCoxoN1makt745rOxeASQAczKo\_9K8IK7i-BZDq6L2bg7Y7Qr20qLQ=="  
 self.org = "Christ"  
 # self.url = "http://localhost:8086"  
 self.url = "http://10.21.70.16:8086"  
 self.bucket = bucket  
 self.client = influxdb\_client.InfluxDBClient(url=self.url, token=self.token, org=self.org)  
 self.write\_api = None  
 self.delete\_api = None  
 self.query\_api = None  
  
 def delete(self, measure, start, stop):  
 if self.delete\_api is None:  
 self.delete\_api = self.client.delete\_api()  
  
 #start = "1970-01-01T00:00:00Z"  
 #stop = "2100-01-01T00:00:00Z"  
 return self.delete\_api.delete(start, stop, "\_measurement="+measure, bucket=self.bucket, org=self.org)  
  
 def write(self, measure, atag, afield):  
 if self.write\_api is None:  
 self.write\_api = self.client.write\_api(write\_options=SYNCHRONOUS)  
  
 point = (Point(measure).tag(atag[0], atag[1]).field(afield[0], afield[1]))  
 return self.write\_api.write(bucket=self.bucket, org=self.org, record=point)  
  
 def query(self, measure, start):  
 if self.query\_api is None:  
 self.query\_api = self.client.query\_api()  
  
 query = f"""from(bucket: "{self.bucket}")  
 |> range(start: {start})  
 |> filter(fn: (r) => r.\_measurement == "{measure}")"""  
 tables = self.query\_api.query(query, org=self.org)  
  
 return tables  
  
 def query\_mean(self, measure, start):  
 if self.query\_api is None:  
 self.query\_api = self.client.query\_api()  
  
 query = f"""from(bucket: "{self.bucket}")  
 |> range(start: {start})  
 |> filter(fn: (r) => r.\_measurement == "{measure}")  
 |> mean()"""  
 tables = self.query\_api.query(query, org=self.org)  
  
 return tables  
  
# Your existing InfluxDB class definition  
  
# Your existing code for fetching outdoor weather data  
def fetch\_weather\_data(city\_name):  
 api\_key = '9e92654b6666b69de295338624d1ac4e'  
 base\_url = f'https://api.openweathermap.org/data/2.5/weather?q={city\_name}&appid={api\_key}&units=metric'  
  
 try:  
 response = requests.get(base\_url)  
 data = response.json()  
  
 if response.status\_code == 200:  
 temperature = data['main']['temp']  
 humidity = data['main']['humidity']  
 return temperature, humidity  
 else:  
 print('Error:', data['message'])  
 return None, None  
  
 except Exception as e:  
 print('Error:', str(e))  
 return None, None  
  
  
# Your existing code for measuring indoor data  
  
  
# MQTT on\_message callback function  
# ... (previous code)  
  
def on\_message(client, userdata, message):  
 if message.topic == "tempTopic":  
 temperature = float(message.payload.decode("utf-8"))  
 print("Received Temperature:", temperature)  
 ts.write("testing", ["location", "indoor"], ["temperature", temperature]) # Indoor temperature  
 elif message.topic == "humiTopic":  
 humidity = float(message.payload.decode("utf-8"))  
 print("Received Humidity:", humidity)  
 ts.write("testing", ["location", "indoor"], ["humidity", humidity]) # Indoor humidity  
  
 # Fetch outdoor weather data  
 city\_name = 'Kolkata'  
 outdoor\_temperature, outdoor\_humidity = fetch\_weather\_data(city\_name)  
  
 if outdoor\_temperature is not None and outdoor\_humidity is not None:  
 print(f'Outdoor Temperature in {city\_name}: {outdoor\_temperature}°C')  
 print(f'Outdoor Humidity in {city\_name}: {outdoor\_humidity}%')  
 ts.write("outdoor\_temperature", ["location", "outdoor"], ["temperature", outdoor\_temperature])  
 ts.write("outdoor\_humidity", ["location", "outdoor"], ["humidity", outdoor\_humidity])  
  
 elif message.topic == "AcStatusTopic":  
 Ac\_Status = int(message.payload.decode("utf-8"))  
 print("Received AC\_Status:", Ac\_Status)  
 ts.write("testing", ["location", "indoor"], ["Ac\_Status", Ac\_Status])  
  
  
  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
  
 measure = "testing" # Specify your measurement name  
  
 ts = InfluxDB(bucket="Weather\_Test\_Data")  
  
 # Set up MQTT client and connect to the broker using the broker address  
 mqtt\_client = mqtt.Client()  
 mqtt\_client.on\_message = on\_message  
 mqtt\_client.connect("10.21.70.16", 1883) # Replace with your MQTT broker address  
 mqtt\_client.subscribe([("tempTopic", 0), ("humiTopic", 0),("AcStatusTopic" , 0)])  
 mqtt\_client.loop\_start()  
  
 try:  
 while True:  
 time.sleep(120) # Sleep for 2 minutes  
 except KeyboardInterrupt:  
 mqtt\_client.disconnect()  
 mqtt\_client.loop\_stop()

**Explanation:**

1. \*\*InfluxDB Class:\*\*

- `\_\_init\_\_(self, bucket="Weather\_Data")`: Initializes the InfluxDB class with default or user-provided values for token, organization (org), URL, and bucket.

- `delete(self, measure, start, stop)`: Deletes data points within the specified time range and measurement from the InfluxDB bucket.

- `write(self, measure, atag, afield)`: Writes data points with the specified measurement, tags, and fields to the InfluxDB bucket.

- `query(self, measure, start)`: Performs a Flux query to retrieve data points of the specified measurement from the InfluxDB bucket within the given time range.

- `query\_mean(self, measure, start)`: Similar to `query`, but calculates the mean value of data points within the specified measurement and time range.

2. \*\*fetch\_weather\_data Function:\*\*

- `fetch\_weather\_data(city\_name)`: Fetches outdoor weather data from OpenWeatherMap API for the provided `city\_name`.

- Constructs the API URL, sends a GET request, and parses the response JSON.

- Returns outdoor temperature and humidity if the API request is successful; otherwise, returns `None, None`.

3. \*\*on\_message Callback Function:\*\*

- `on\_message(client, userdata, message)`: Callback function triggered when an MQTT message is received.

- Handles messages for "tempTopic," "humiTopic," and "AcStatusTopic."

- Decodes message payloads, updates InfluxDB with indoor temperature, humidity, and AC status data, and fetches and writes outdoor weather data.

- Handles exceptions and error cases during data retrieval and writing.

4. \*\*Main Execution Block:\*\*

- Initializes the `InfluxDB` class instance `ts` with the specified bucket name ("Weather\_Data").

- Sets up the MQTT client (`mqtt\_client`) and connects it to the specified MQTT broker address and port.

- Subscribes to MQTT topics ("tempTopic," "humiTopic," and "AcStatusTopic") with QoS level 0.

- Starts the MQTT client's loop to listen for incoming MQTT messages.

5. \*\*Main Loop:\*\*

- The script enters an infinite loop (`while True`) for continuous execution.

- It periodically sleeps for 2 minutes (`time.sleep(120)`) to manage resource usage.

- The loop continues until a KeyboardInterrupt (`Ctrl+C`) is detected.

6. \*\*Exception Handling:\*\*

- Inside the `except KeyboardInterrupt` block, the script disconnects the MQTT client (`mqtt\_client.disconnect()`) and stops its loop (`mqtt\_client.loop\_stop()`).

Overall, the code establishes a connection to an MQTT broker, subscribes to MQTT topics, processes received messages to update the InfluxDB database with temperature, humidity, and AC status data, fetches outdoor weather data, and maintains a loop for continuous execution until manually interrupted.

**Fetching the data from influxdb for ChatGPT recommendation:**

**Code :**

import influxdb\_client  
from influxdb\_client import InfluxDBClient, Point, WritePrecision  
from influxdb\_client.client.write\_api import SYNCHRONOUS  
from datetime import timedelta, datetime  
import openai  
import re  
import speech\_recognition as sr  
import pyttsx3  
import telebot  
  
  
class InfluxDB:  
 def \_\_init\_\_(self, bucket="Weather\_Test\_Data"):  
 self.token = "4S-CyOmTPUhSgUiUSf0A8Fn31keTgErGXCoxoN1makt745rOxeASQAczKo\_9K8IK7i-BZDq6L2bg7Y7Qr20qLQ=="  
 self.org = "Christ"  
 # self.url = "http://localhost:8086"  
 self.url = "http://10.21.70.16:8086"  
 self.bucket = bucket  
 self.client = influxdb\_client.InfluxDBClient(url=self.url, token=self.token, org=self.org)  
 self.write\_api = None  
 self.delete\_api = None  
 self.query\_api = None  
  
 def delete(self, measure, start, stop):  
 if self.delete\_api is None:  
 self.delete\_api = self.client.delete\_api()  
  
 #start = "1970-01-01T00:00:00Z"  
 #stop = "2100-01-01T00:00:00Z"  
 return self.delete\_api.delete(start, stop, "\_measurement="+measure, bucket=self.bucket, org=self.org)  
  
 def write(self, measure, atag, afield):  
 if self.write\_api is None:  
 self.write\_api = self.client.write\_api(write\_options=SYNCHRONOUS)  
  
 point = (Point(measure).tag(atag[0], atag[1]).field(afield[0], afield[1]))  
 return self.write\_api.write(bucket=self.bucket, org=self.org, record=point)  
  
 # def query(self, measure, start):  
 # if self.query\_api is None:  
 # self.query\_api = self.client.query\_api()  
 #  
 # start = "2023-07-29T00:00:00Z"  
 #  
 # query = f"""from(bucket: "{self.bucket}")  
 # |> range(start: {start})  
 # |> filter(fn: (r) => r.\_measurement == "{measure}")"""  
 # tables = self.query\_api.query(query, org=self.org)  
 #  
 # return tables  
  
 def query\_mean(self, measure, start):  
 if self.query\_api is None:  
 self.query\_api = self.client.query\_api()  
  
 query = f"""from(bucket: "{self.bucket}")  
 |> range(start: {start})  
 |> filter(fn: (r) => r.\_measurement == "{measure}")  
 |> mean()"""  
 tables = self.query\_api.query(query, org=self.org)  
  
 return tables  
  
  
 def get\_indoor\_humidity(self, start\_date, end\_date):  
 query = f"""from(bucket: "{self.bucket}")  
 |> range(start: {start\_date.strftime('%Y-%m-%dT%H:%M:%SZ')}, stop: {end\_date.strftime('%Y-%m-%dT%H:%M:%SZ')})  
 |> filter(fn: (r) => r.\_measurement == "indoor\_humidity" and r.\_field == "value" and r.location == "indoor")  
 |> window(every: 1d)  
 |> aggregateWindow(every: 1h, fn: mean, createEmpty: false)"""  
  
 result = self.query(query)  
  
 humidity\_data\_by\_day = {day: [] for day in range(1, 4)}  
  
 for table in result:  
 for record in table.records:  
 timestamp = record["\_time"]  
 day = (timestamp.date() - start\_date.date()).days + 1  
 humidity = record["\_value"]  
 humidity\_data\_by\_day[day].append(humidity)  
  
 return humidity\_data\_by\_day  
  
 def get\_outdoor\_humidity(self, start\_date, end\_date):  
 query = f"""from(bucket: "{self.bucket}")  
 |> range(start: {start\_date.strftime('%Y-%m-%dT%H:%M:%SZ')}, stop: {end\_date.strftime('%Y-%m-%dT%H:%M:%SZ')})  
 |> filter(fn: (r) => r.\_measurement == "testing" and r.\_field == "humidity" and r.location == "outdoor")  
 |> window(every: 1d)  
 |> aggregateWindow(every: 1h, fn: mean, createEmpty: false) """  
  
 result = self.query(query)  
  
 humidity\_data\_by\_day = {day: [] for day in range(1, 4)}  
  
 for table in result:  
 for record in table.records:  
 try:  
 timestamp = record["\_time"]  
 day = (timestamp.date() - start\_date.date()).days + 1  
 humidity = record["\_value"]  
 humidity\_data\_by\_day[day].append(humidity)  
 except:  
 continue  
  
 return humidity\_data\_by\_day  
  
 def get\_outdoor\_temperature(self, start\_date, end\_date):  
 query = f"""from(bucket: "{self.bucket}")  
 |> range(start: {start\_date.strftime('%Y-%m-%dT%H:%M:%SZ')}, stop: {end\_date.strftime('%Y-%m-%dT%H:%M:%SZ')})  
 |> filter(fn: (r) => r.\_measurement == "testing" and r.\_field == "temperature" and r.location == "outdoor")  
 |> window(every: 1d)  
 |> aggregateWindow(every: 1h, fn: mean, createEmpty: false)"""  
  
 result = self.query(query)  
  
 temperature\_data\_by\_day = {day: [] for day in range(1, 4)}  
  
 for table in result:  
 for record in table.records:  
 try:  
 timestamp = record["\_time"]  
 day = (timestamp.date() - start\_date.date()).days + 1  
 temperature = record["\_value"]  
 temperature\_data\_by\_day[day].append(temperature)  
 except:  
 continue  
 return temperature\_data\_by\_day  
  
 def get\_indoor\_temperature(self, start\_date, end\_date):  
 query = f"""from(bucket: "{self.bucket}")  
 |> range(start: {start\_date.strftime('%Y-%m-%dT%H:%M:%SZ')}, stop: {end\_date.strftime('%Y-%m-%dT%H:%M:%SZ')})  
 |> filter(fn: (r) => r.\_measurement == "testing" and r.\_field == "temperature" and r.location == "indoor")  
 |> window(every: 1d)  
 |> aggregateWindow(every: 1h, fn: mean, createEmpty: false)"""  
  
 result = self.query(query)  
  
 temperature\_data\_by\_day = {day: [] for day in range(1, 4)}  
  
 for table in result:  
 for record in table.records:  
 try:  
 timestamp = record["\_time"]  
 day = (timestamp.date() - start\_date.date()).days + 1  
 temperature = record["\_value"]  
 temperature\_data\_by\_day[day].append(temperature)  
 except:  
 continue  
  
 return temperature\_data\_by\_day  
  
 def get\_ac\_status(self, start\_date, end\_date):  
 query = f"""from(bucket: "{self.bucket}")  
 |> range(start: {start\_date.strftime('%Y-%m-%dT%H:%M:%SZ')}, stop: {end\_date.strftime('%Y-%m-%dT%H:%M:%SZ')})  
 |> filter(fn: (r) => r.\_measurement == "testing" and r.\_field == "AC\_Status" and r.location == "indoor")  
 |> window(every: 1d)  
 |> aggregateWindow(every: 1h, fn: mean, createEmpty: false)"""  
  
 result = self.query(query)  
  
 ac\_status\_data\_by\_day = {day: [] for day in range(1, 4)}  
  
 for table in result:  
 for record in table.records:  
 try:  
 timestamp = record["\_time"]  
 day = (timestamp.date() - start\_date.date()).days + 1  
 ac\_status = list(map(round,record["\_value"]))  
 ac\_status\_data\_by\_day[day].append(ac\_status)  
 except:  
 continue  
  
 return ac\_status\_data\_by\_day  
  
 def get\_time\_data(self):  
  
  
 time\_data\_by\_day = ['00:00:00', '01:00:00', '02:00:00', '03:00:00', '04:00:00', '05:00:00', '06:00:00', '07:00:00', '08:00:00', '09:00:00', '10:00:00', '11:00:00', '12:00:00', '13:00:00', '14:00:00', '15:00:00', '16:00:00', '17:00:00', '18:00:00', '19:00:00', '20:00:00', '21:00:00', '22:00:00', '23:00:00']  
  
  
 return time\_data\_by\_day  
  
  
  
 def query(self, query):  
 if self.query\_api is None:  
 self.query\_api = self.client.query\_api()  
  
 return self.query\_api.query(query, org=self.org)  
  
 def get\_formatted\_data(self, measurement, field, location, start\_date, end\_date):  
 query = f"""from(bucket: "{self.bucket}")  
 |> range(start: {start\_date.strftime('%Y-%m-%dT%H:%M:%SZ')}, stop: {end\_date.strftime('%Y-%m-%dT%H:%M:%SZ')})  
 |> filter(fn: (r) => r.\_measurement == "{measurement}" and r.\_field == "{field}" and r.location == "{location}")  
 """  
  
 result = self.query(query)  
  
 formatted\_data\_by\_day = {}  
  
 for table in result:  
 for record in table.records:  
 timestamp = record["\_time"]  
 day = (timestamp.date() - start\_date.date()).days + 1  
  
 if day not in formatted\_data\_by\_day:  
 formatted\_data\_by\_day[day] = {  
 "time": [],  
 "indoor Humidity": [],  
 "Outdoor Humidity Data": [],  
 "Indoor Temperature Data": [],  
 "Outdoor Temperature Data": [],  
 "AC\_Status Data": []  
 }  
  
 formatted\_data\_by\_day[day]["time"].append(timestamp.strftime("%H:%M:%S"))  
 formatted\_data\_by\_day[day]["indoor Humidity"].append(record["\_value"])  
   
 return formatted\_data\_by\_day  
  
  
token = "4S-CyOmTPUhSgUiUSf0A8Fn31keTgErGXCoxoN1makt745rOxeASQAczKo\_9K8IK7i-BZDq6L2bg7Y7Qr20qLQ==" # Replace with your InfluxDB token # Replace with your InfluxDB token  
bucket = "Weather\_Test\_Data"  
  
# Calculate the start date (last three days from July 31, 2023)  
end\_date = datetime(2023, 8, 1)  
start\_date = end\_date - timedelta(days=3)  
  
# Create an InfluxDB instance without passing the 'token' argument  
influx\_db = InfluxDB()  
  
# Fetch indoor humidity data using the class method  
indoor\_humidity\_data = influx\_db.get\_indoor\_humidity(start\_date, end\_date)  
  
# Fetch outdoor humidity data using the class method  
outdoor\_humidity\_data = influx\_db.get\_outdoor\_humidity(start\_date, end\_date)  
  
# Fetch outdoor temperature data using the class method  
outdoor\_temperature\_data = influx\_db.get\_outdoor\_temperature(start\_date, end\_date)  
  
# Fetch indoor temperature data using the class method  
indoor\_temperature\_data = influx\_db.get\_indoor\_temperature(start\_date, end\_date)  
  
# Fetch AC\_Status data using the class method  
ac\_status\_data = influx\_db.get\_ac\_status(start\_date, end\_date)  
  
# Fetch time data using the class method  
time\_data = influx\_db.get\_time\_data()  
  
  
# Fetch and format data using the class method  
formatted\_data = influx\_db.get\_formatted\_data("testing", "humidity", "indoor", start\_date, end\_date)  
  
# Add outdoor humidity data  
formatted\_data\_outdoor\_humidity = influx\_db.get\_formatted\_data("testing", "humidity", "outdoor", start\_date, end\_date)  
for day, data in formatted\_data\_outdoor\_humidity.items():  
 formatted\_data[day]["Outdoor Humidity Data"] = data["indoor Humidity"]  
  
# Add indoor temperature data  
formatted\_data\_indoor\_temp = influx\_db.get\_formatted\_data("testing", "temperature", "indoor", start\_date, end\_date)  
for day, data in formatted\_data\_indoor\_temp.items():  
 formatted\_data[day]["Indoor Temperature Data"] = data["indoor Humidity"]  
  
# Add outdoor temperature data  
formatted\_data\_outdoor\_temp = influx\_db.get\_formatted\_data("testing", "temperature", "outdoor", start\_date, end\_date)  
for day, data in formatted\_data\_outdoor\_temp.items():  
 formatted\_data[day]["Outdoor Temperature Data"] = data["indoor Humidity"]  
  
# Add AC status data  
formatted\_data\_ac\_status = influx\_db.get\_formatted\_data("testing", "AC\_Status", "indoor", start\_date, end\_date)  
for day, data in formatted\_data\_ac\_status.items():  
 formatted\_data[day]["AC\_Status Data"] = data["indoor Humidity"]  
  
# Print the formatted data for each day  
for day, data in formatted\_data.items():  
 print(f"Day {day} Data:")  
 print(data)  
 print()  
  
# ... (previous code)  
  
# Create a dictionary to store the combined data for each day  
combined\_data = {}  
  
  
# Combine all the data into the desired format  
for day, data in formatted\_data.items():  
 combined\_data[day] = {  
 "time": data["time"],  
 "indoor Humidity Data": data["indoor Humidity"],  
 "Outdoor Humidity Data": formatted\_data\_outdoor\_humidity[day]["indoor Humidity"],  
 "Indoor Temperature Data": formatted\_data\_indoor\_temp[day]["indoor Humidity"],  
 "Outdoor Temperature Data": formatted\_data\_outdoor\_temp[day]["indoor Humidity"],  
 "AC\_Status Data": formatted\_data\_ac\_status[day]["indoor Humidity"]  
 }  
  
# Print the combined data dictionary  
print(combined\_data)  
  
# Set your OpenAI API key here  
openai.api\_key = "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"  
  
# Define the data  
data = combined\_data  
  
  
# Function to recognize speech using the microphone  
def recognize\_speech():  
 recognizer = sr.Recognizer()  
  
 with sr.Microphone() as source:  
 print("Say something:")  
 audio = recognizer.listen(source)  
  
 try:  
 recognized\_text = recognizer.recognize\_google(audio)  
 print("You said:", recognized\_text)  
 return recognized\_text  
 except sr.UnknownValueError:  
 print("Speech Recognition could not understand the audio.")  
 except sr.RequestError as e:  
 print(f"Could not request results from Google Web Speech Recognition service; {e}")  
 except Exception as e:  
 print("Unknown error:", e)  
 return None  
  
  
# Function to analyze data using OpenAI  
def analyze\_data(question):  
 try:  
 # Check if the recognized text starts with "System"  
 if question.lower().startswith("system"):  
 # prompt = f"""System: Analyse the following last three days time series data which is json format and tell me the time period based on the question. Give answer like as recommendation  
 # - the (suitable or perfect or right or recommend ) time to turn on the AC (from when (AM or PM )to when(AM or PM) ) and give recomendation why AC should be turn on in creative way  
 # - do not give strange time interval like at night timing from 10 PM to 5AM dont recomend this time period  
 #  
 # Data: {data}  
 #  
 # User: {question}  
 # """  
  
 prompt =f""" Analyse the following last three days data which is in json format and recommend a time period when the ac should be turn on in morning and afternoo  
 - give time period like (from when (AM or PM )to when(AM or PM) ) for each morning and afternoon  
 - give reasons also  
 Data: {data}  
   
 User: {question}  
 """  
  
 response = openai.Completion.create(  
 engine="text-davinci-003",  
 prompt=prompt,  
 max\_tokens=150,  
 temperature=0.7,  
 )  
  
 answer = response.choices[0].text.strip()  
  
 time\_match = re.search(r'\d{1,2}:\d{2} [APap][Mm]', answer)  
 if time\_match:  
 extracted\_time = time\_match.group()  
 answer = answer.replace("[time]", extracted\_time)  
  
 return answer  
 else:  
 return "Please start your question with 'System'."  
  
 except Exception as e:  
 return f"Error: {e}"  
  
  
# Function to play TTS audio  
# Function to convert text to speech using pyttsx3  
def convert\_to\_speech(text):  
 engine = pyttsx3.init()  
 # Skip speaking the "Answer" part  
 if text.lower().startswith("answer"):  
 text = text.split(" ", 1)[1]  
 elif text.lower().startswith("system"):  
 text = text.split(" ", 1)[1]  
 engine.say(text)  
 engine.runAndWait()  
  
  
  
bot\_token = '6505007326:AAFsm0gFmkccJR8zO3LAaoO0zwKUW\_Fv2BE'  
bot = telebot.TeleBot(bot\_token)  
  
# Function to send a message  
def send\_telegram\_message(message):  
 # Replace with the actual chat ID where you want to send the message  
 bot\_chat\_id = '5906755016'  
  
  
 # Check the message before sending  
 if not message.strip().lower().startswith("please start your question with 'system'"):  
 # Send the message  
 bot.send\_message(chat\_id=bot\_chat\_id, text=message)  
  
  
  
# Main program  
def main():  
 while True:  
 question = recognize\_speech()  
 if question:  
 answer = analyze\_data(question)  
 print(f"Q: {question}\nA: {answer}")  
 convert\_to\_speech(answer)  
 if answer.lower().startswith("answer:"):  
 answer = answer.split(" ", 1)[1]  
 # Send the message using the function  
 send\_telegram\_message(answer)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Explanation:**

1. **InfluxDB Class (Partial):**
   * **\_\_init\_\_(self, bucket="Weather\_Data")**: Constructor initializes the InfluxDB client with the provided token, organization, URL, and bucket.
   * **delete(self, measure, start, stop)**: Deletes data points from the specified measurement within the given time range.
   * **write(self, measure, atag, afield)**: Writes data points to the specified measurement with provided tags and fields.
   * **query\_mean(self, measure, start)**: Executes a mean aggregation query on the specified measurement within the given time range.
2. **Additional InfluxDB Methods:**
   * **get\_indoor\_humidity**, **get\_outdoor\_humidity**, **get\_outdoor\_temperature**, **get\_indoor\_temperature**, **get\_ac\_status**, **get\_time\_data**, **get\_formatted\_data**: These methods retrieve specific data from the InfluxDB database within a specified time range and organize it by days.
3. **Data Collection:**
   * Fetches indoor and outdoor humidity, indoor and outdoor temperature, and AC status data using the **InfluxDB** class methods.
   * Retrieves data for the last three days.
4. **OpenAI and Analysis Functions:**
   * **analyze\_data(question)**: Utilizes OpenAI's GPT-3 model to analyze the user's question and generate a response.
   * Creates a prompt incorporating the question and data, asking for a suitable time to turn on the AC with reasons.
5. **Voice Recognition and TTS:**
   * **recognize\_speech()**: Uses the microphone to capture user speech, converts it to text using Google Web Speech Recognition.
   * **convert\_to\_speech(text)**: Converts text to speech using the **pyttsx3** library for audio output.
6. **Telegram Integration:**
   * **send\_telegram\_message(message)**: Sends a message to a specified Telegram chat using the **telebot** library.
7. **Main Program:**
   * Initializes a loop that continuously listens for user speech input.
   * If user input is detected, it is recognized, analyzed using OpenAI, and the response is converted to speech.
   * The response is also sent to the specified Telegram chat using the **send\_telegram\_message** function.

**Arduino Source code :**

**Code:**

#include <Ethernet.h>

#include <Wire.h>

#include <SoftWire.h>

#include <DHT.h>

#include <PubSubClient.h>

#include <ArduinoJson.h>

#include "HardwareSerial.h"

// DHT sensor setup

#define DHTPIN 15     // Pin where the DHT11 sensor is connected

#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);

// AC current setup

#define ACPIN 14

#define ACON\_THRESHOLD 200

// Update these with values suitable for your network.

byte mac[]    = {  0xDE, 0xED, 0xBA, 78, 0xFE, 0xED };

IPAddress ip(172, 16, 0, 100);

//IPAddress server(44, 195, 202, 69);

IPAddress server(10, 21, 70, 16);

void callback(char\* topic, byte\* payload, unsigned int length) {

  // Handle incoming MQTT messages based on the topic if needed

}

EthernetClient ethClient;

PubSubClient client(ethClient);

void reconnect() {

  // Loop until we're reconnected

  while (!client.connected()) {

    Serial.print("Attempting MQTT connection...");

    // Attempt to connect

    if (client.connect("arduinoClient78")) {

      Serial.println("connected");

      // Once connected, resubscribe to the topics

      client.subscribe("tempTopic");

      client.subscribe("humiTopic");

      client.subscribe("ACStatusTopic");

    } else {

      Serial.print("failed, rc=");

      Serial.print(client.state());

      Serial.println(" try again in 5 seconds");

      // Wait 5 seconds before retrying

      delay(5000);

    }

  }

}

void setup() {

    // Open serial communications and wait for port to open:

  Serial3.setRx(PC11);

  Serial3.setTx(PC10);

  delay(50);

  Serial.begin(115200);

  //Ethernet.init(17);

  dht.begin();

  client.setServer(server, 1883);

  client.setCallback(callback);

  Ethernet.begin(mac);

  // Allow the hardware to sort itself out

  delay(1500);

}

void loop() {

  if (!client.connected()) {

    reconnect();

  }

  client.loop();

  // Read temperature and humidity from the sensor

  float humidity = dht.readHumidity();

  float temperature = dht.readTemperature();

  if (isnan(humidity) || isnan(temperature)) {

    Serial.println("Failed to read from DHT sensor!");

    return;

  }

  Serial.print("Temperature: ");

  Serial.print(temperature);

  Serial.print(" °C\t");

  Serial.print("Humidity: ");

  Serial.print(humidity);

  Serial.println(" %");

  // Convert readings to strings

  String tempStr = String(temperature);

  String humStr = String(humidity);

  // Publish temperature and humidity readings to MQTT topics

  client.publish("tempTopic", tempStr.c\_str());

  client.publish("humiTopic", humStr.c\_str());

  // Publish AC status to MQTT topics

  int combinedCurrent = analogRead(ACPIN);

  Serial.println(combinedCurrent);

  if (combinedCurrent > ACON\_THRESHOLD) {

    Serial.println("AC Status: ON");

    client.publish("ACStatusTopic", "1");

  } else {

    Serial.println("AC Status: OFF");

    client.publish("ACStatusTopic", "0");

  }

  // Wait for 2 minue before taking the next reading

  //delay(120000);

  delay(6000);

}

**Explanation:**

1. **Callback Function (callback):**
   * This function is intended to handle incoming MQTT messages based on the subscribed topics.
   * However, the actual handling logic is not implemented in the code snippet provided.
2. **Reconnection Function (reconnect):**
   * This function is responsible for attempting to reconnect to the MQTT broker if the client is not currently connected.
   * It goes into a loop until the client successfully reconnects.
   * If reconnection fails, it prints the client's state and waits for 5 seconds before trying again.
3. **Setup Function (setup):**
   * Initializes the hardware and configurations during the setup phase of the Arduino sketch.
   * Configures serial communication with the Serial3 interface.
   * Begins serial communication with a baud rate of 115200.
   * Initializes the DHT sensor to read data from the specified pin and of the specified type (DHT11).
   * Sets up the PubSubClient with the MQTT server and callback function.
   * Initializes the Ethernet communication and allows time for the Ethernet hardware to initialize (1500 ms).
4. **Main Loop Function (loop):**
   * The main loop of the Arduino sketch, which continuously executes after the setup phase.
   * Checks if the MQTT client is not connected, and if so, attempts to reconnect using the **reconnect** function.
   * Executes the PubSubClient's loop function, which handles MQTT communication.
   * Reads the temperature and humidity data from the DHT sensor.
   * Checks if the sensor readings are valid (not NaN), and if they are, publishes the temperature and humidity readings to their respective MQTT topics ("tempTopic" and "humiTopic").
   * Reads the combined current from an analog pin (ACPIN), which is presumably connected to an AC current sensor.
   * Compares the combined current value with a predefined threshold (ACON\_THRESHOLD) to determine whether the AC is ON or OFF.
   * Prints the AC status and publishes it to the "ACStatusTopic" MQTT topic as "1" (ON) or "0" (OFF).
   * Adds a delay of 6 seconds before the next iteration of the loop.

Overall, this code combines reading data from a DHT11 sensor, monitoring AC current, and publishing these readings to MQTT topics using an Ethernet connection.