**[AIoT]Smart Mood Light with ChatGPT + VoiceRecognition + TTS**

**CODE EXPLAINITION:**

**Python Explanation:**

import paho.mqtt.client as mqtt

import openai

import re

import speech\_recognition as sr

import threading

import pyttsx3

# Set up MQTT Broker details

broker\_address = "broker.emqx.io"

broker\_port = 1883

voice\_command\_topic = "voice\_command\_topic"

response\_topic = "response\_topic"

rgb\_values\_topic = "rgb\_values\_topic"

# Set up OpenAI API key

openai.api\_key = "sk-b7vcU0E2ekDx5ga0\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*KoPb4L60"

# Function to recognize speech using the microphone

def recognize\_speech():

# ... (explained below)

# Function to process the AI model (OpenAI GPT-3)

def process\_ai\_model(voice\_command):

# ... (explained below)

# Function to extract color name from the generated text

def extract\_color\_name(generated\_text):

# ... (explained below)

# Function to generate RGB values based on the color name

def generate\_rgb\_values(generated\_text):

# ... (explained below)

# MQTT client callbacks

def on\_connect(client, userdata, flags, rc):

# ... (explained below)

def text\_to\_speech(text):

# ... (explained below)

# Function to perform speech recognition in a separate thread

def perform\_speech\_recognition():

# ... (explained below)

# Create an MQTT client instance

client = mqtt.Client()

# Set up the callback functions

client.on\_connect = on\_connect

# No need to set client.on\_message here, it will be handled later.

# Connect to the MQTT broker

client.connect(broker\_address, broker\_port)

# Start the MQTT loop to maintain the connection and handle callbacks

client.loop\_start()

# Start the speech recognition thread

speech\_thread = threading.Thread(target=perform\_speech\_recognition)

speech\_thread.daemon = True # The thread will terminate when the main program ends.

speech\_thread.start()

# Loop indefinitely to keep the program running

while True:

pass

**Explanation:**

* The code starts by importing necessary libraries: paho.mqtt.client for MQTT communication, openai for the OpenAI API, re for regular expressions, speech\_recognition for speech recognition, threading for creating separate threads, and pyttsx3 for text-to-speech functionality.
* The code sets up the MQTT broker details, such as the broker's address, port, and topic names.
* recognize\_speech() is a function that uses the microphone to recognize speech using the Google Web Speech Recognition service. It returns the recognized text.
* process\_ai\_model(voice\_command) is a function that takes the recognized voice command as input and uses OpenAI's GPT-3 model to process it and generate an AI response based on the given prompt.
* extract\_color\_name(generated\_text) is a function that extracts the color name from the generated AI text using regular expressions. It matches color names and their variations in the generated text.
* generate\_rgb\_values(generated\_text) is a function that takes the generated AI text as input and uses the extracted color name to map it to predefined RGB values for different colors.
* on\_connect(client, userdata, flags, rc) is a callback function that is called when the MQTT client connects to the broker. It subscribes to the voice\_command\_topic.
* text\_to\_speech(text) is a function that uses the pyttsx3 library to convert text into speech and play it.
* perform\_speech\_recognition() is a function that runs in a separate thread and continuously performs speech recognition using the recognize\_speech() function. It processes the voice command using the AI model, generates an AI response, extracts the color name, generates RGB values, and publishes the AI response and RGB values to their respective topics using MQTT.
* The code creates an MQTT client instance and sets up the necessary callbacks.
* It connects the MQTT client to the broker and starts the MQTT loop to maintain the connection and handle callbacks.
* It starts the perform\_speech\_recognition() function in a separate thread to perform speech recognition continuously.
* The main program enters an infinite loop to keep it running, and the speech recognition thread runs in the background to handle voice commands and generate AI responses.

**Python Source Code:**

import paho.mqtt.client as mqtt

import openai

import re

import speech\_recognition as sr

import threading

import pyttsx3 # Import the pyttsx3 library

# Set up MQTT Broker details

broker\_address = "broker.emqx.io"

broker\_port = 1883

voice\_command\_topic = "voice\_command\_topic"

response\_topic = "response\_topic"

# MQTT topic for RGB values

rgb\_values\_topic = "rgb\_values\_topic"

# Set up OpenAI API key

openai.api\_key = "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"

# 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:

print("Unknow error")

# Function to process the AI model (OpenAI GPT-3)

def process\_ai\_model(voice\_command):

# Prepare the prompt for the OpenAI API

prompt = f"""Convert the given text to a command among the following options:

- Turn off the light

- Turn on the light

- Change the light to a specific color

Text: {voice\_command}

Conditions:

1. Find a specific color based on the text even if color is not directly mentioned.

2. If the text contains a specific color, choose the command "Turn on [color] light".

3. If the text contains the word "party" or "rainbow," choose the command "Turn on all hue lights."

4. If the text mentions colors indirectly (e.g., sky-like, ocean-like, ocean blue , sea green ,midnight blue,navy blue ,navi blue ,etc .),

map the keywords like (e.g sky ,ocean, sea, midnight,navy, navi etc) to colors and choose the closest matching color. Then, use the command "

Turn on [keyword] [closest\_color] or [colour] light."

5. If the text contains the word "dim" and a color, choose the command "Turn on light [color] with reduced intensity."

6. If the text does not contain any color, choose either "Turn off the light" or "Turn on the light" command.

"""

# Call the OpenAI API to generate the AI response

respond = openai.Completion.create(

model="text-davinci-003",

prompt=prompt,

temperature=0,

max\_tokens=100,

top\_p=1,

frequency\_penalty=0.2,

presence\_penalty=0

)

# Extract the generated text from the response

generated\_text = respond.choices[0].text.strip()

# List of action-related keywords

action\_keywords = ["turn on", "turn off", "change", "set"]

# Regular expression patterns to match color names

color\_patterns = {

"red": r"\bred\b",

"green": r"\bgreen\b",

"blue": r"\bblue\b",

"magenta": r"\bmagenta\b",

"yellow": r"\byellow\b",

"orange": r"\borange\b",

"pink": r"\bpink\b",

"light green": r"\blight\s\*green\b",

"sky blue": r"\bsky\s\*blue\b",

"purple": r"\bpurple\b",

"cyan": r"\bcyan\b",

"gold": r"\bgold\b",

"brown": r"\bbrown\b",

"teal": r"\bteal\b",

"lavender": r"\blavender\b",

"indigo": r"\bindigo\b",

"maroon": r"\bmaroon\b",

"lime": r"\blime\b",

"olive": r"\bolive\b",

"aqua": r"\baqua\b",

"midnight blue": r"\bmidnight\s\*blue\b",

"ocean blue": r"\bocean\s\*blue\b",

"sea green": r"\bsea\s\*green\b",

"navy blue": r"\bnavy\s\*blue\b",

"crimson" : r"\bcrimson\s\*red\b"

# ... (other color patterns as before) ...

}

# Check if the generated\_text contains any action-related keyword

if any(keyword in generated\_text.lower() for keyword in action\_keywords):

return generated\_text

# If no action-related keyword is found, extract color name using regular expressions

extracted\_color = None

for color\_name, pattern in color\_patterns.items():

if re.search(pattern, generated\_text, re.IGNORECASE):

extracted\_color = color\_name

break

# If no color is extracted and no action-related keyword is found, return the default action (turn off the light)

if not extracted\_color:

return "Turn off the light"

# If color is extracted, return the response

return generated\_text

# Function to extract color name from the generated text

def extract\_color\_name(generated\_text):

# Define a list of color names and their variations

color\_names = {

"red": ["red"],

"green": ["green"],

"blue": ["blue"],

"magenta": ["magenta"],

"yellow": ["yellow"],

"orange": ["orange"],

"pink": ["pink"],

"light green": ["light green", "lightgreen", "light-green"],

"sky blue": ["sky blue", "skyblue", "sky-blue", "light blue"],

"midnight blue": ["midnightblue" ,"midnight-blue" ,"midnightlikeblue" ,"midnight blue"],

"aqua": ["aqua"],

"cyan": ["cyan"],

"lavender": ["lavender"],

"indigo": ["indigo"],

"maroon": ["maroon"],

"lime": ["lime"],

"gold": ["gold"],

"teal": ["teal"],

"purple": ["purple"],

"olive": ["olive"],

"ocean blue": ["oceanblue", "ocean" ,"ocean blue" ,"ocean view"],

"sea green": ["seagreen","sea green" ,"sea-green"],

"navy blue": ["navy blue" ,"navi blue" ,"navyblue","navy" ,"navi"],

"crimson": ["crimson" , "crimson red" , "crimsonred"]

# Add more colors and their variations as needed

}

color\_name = None

for name, variations in color\_names.items():

for variation in variations:

if re.search(rf"\b{re.escape(variation)}\b", generated\_text, re.IGNORECASE):

color\_name = name

break

# Additional logic to handle "dim" for all colors

if "dim" in generated\_text.lower() and color\_name:

color\_name = "light " + color\_name

print("Extracted color name:", color\_name)

return color\_name

# Function to generate RGB values based on the color name

def generate\_rgb\_values(generated\_text):

# Extract the color name from the generated text

color\_name = extract\_color\_name(generated\_text)

# Dictionary with predefined color names and their RGB values

color\_values = {

"red": (255, 0, 0),

"green": (0, 255, 0),

"blue": (0, 0, 255),

"magenta": (255, 0, 255),

"yellow": (255, 255, 0),

"orange": (255, 69, 0),

"pink": (255, 80, 80),

"light green": (80, 250, 50),

"sky blue": (135, 206, 250),

"purple": (128, 0, 128),

"cyan": (0, 139, 139),

"gold": (255, 215, 0),

"brown": (165, 42, 42),

"teal": (0, 128, 128),

"lavender": (230, 230, 250),

"indigo": (75, 0, 130),

"maroon": (128, 0, 0),

"lime": (0, 128, 0),

"olive": (128, 128, 0),

"aqua": (0, 255, 255),

"midnight blue": (25, 25, 112),

"ocean blue": (0, 119, 190),

"sea green": (20, 255, 105),

"navy blue": (0,0,128),

"crimson": (220,20,60)

# Add more colors as needed

}

# Check if the color\_name exists in the dictionary

if color\_name and color\_name.lower() in color\_values:

return color\_values[color\_name.lower()]

elif "turn on" in generated\_text.lower():

# Default to white color (255, 255, 255) if "turn on" is mentioned

return (255, 255, 255)

elif "turn off" in generated\_text.lower():

# Default to black color (0, 0, 0) if "turn off" is mentioned

return (0, 0, 0)

else:

# Default to black color (0, 0, 0) for other cases

return (0, 0, 0)

# MQTT client callbacks

def on\_connect(client, userdata, flags, rc):

print("Connected with result code " + str(rc))

client.subscribe(voice\_command\_topic)

def text\_to\_speech(text):

# Initialize the text-to-speech engine

engine = pyttsx3.init()

# Set properties (optional)

engine.setProperty("rate", 150) # Speed of speech (words per minute)

engine.setProperty("volume", 0.9) # Volume level (0.0 to 1.0)

# Preprocess the text to remove "Answer:" or any other prefix

# You can add more prefixes to remove if needed

prefixes\_to\_remove = ["Answer:"]

for prefix in prefixes\_to\_remove:

if text.startswith(prefix):

text = text[len(prefix):].strip()

# Convert text to speech and play it

engine.say(text)

engine.runAndWait()

# Function to perform speech recognition in a separate thread

def perform\_speech\_recognition():

while True:

voice\_command = recognize\_speech()

if voice\_command:

print("Received voice command:", voice\_command)

first\_word = voice\_command.split()[0].lower()

# Check if the first word matches the "special\_word"

if first\_word == "chat": # Replace "special\_word" with the desired word

# Remove the special word from the voice command

voice\_command\_without\_special\_word = ' '.join(voice\_command.split()[1:])

# If the text is empty after removing the special word, go for speech recognition again

if not voice\_command\_without\_special\_word.strip():

print("No valid command. Trying speech recognition again.")

text\_to\_speech("No valid command. Please try again.") # Provide voice feedback

continue

# Process the voice command using the AI model (e.g., ChatGPT)

response = process\_ai\_model(voice\_command\_without\_special\_word)

print("AI Response:", response)

# Provide voice feedback for AI response

text\_to\_speech(response)

# Extract color name from the generated text

color\_name = extract\_color\_name(response)

print("Extracted color name:", color\_name)

# Generate RGB values based on the color name

rgb\_values = generate\_rgb\_values(response)

print("Generated RGB values:", rgb\_values)

# Publish the AI-generated response to the response topic

client.publish(response\_topic, response)

# Publish RGB values to the RGB values topic

client.publish(rgb\_values\_topic, f"{rgb\_values[0]},{rgb\_values[1]},{rgb\_values[2]}")

else:

print("Voice command does not contain the special word. Trying speech recognition again.")

text\_to\_speech("No valid command. Please try again.") # Provide voice feedback

# Create an MQTT client instance

client = mqtt.Client()

# Set up the callback functions

client.on\_connect = on\_connect

# No need to set client.on\_message here, it will be handled later.

# Connect to the MQTT broker

client.connect(broker\_address, broker\_port)

# Start the MQTT loop to maintain the connection and handle callbacks

client.loop\_start()

# Start the speech recognition thread

speech\_thread = threading.Thread(target=perform\_speech\_recognition)

speech\_thread.daemon = True # The thread will terminate when the main program ends.

speech\_thread.start()

# Loop indefinitely to keep the program running

while True:

pass

**Using Arduino connecting MQTT with Wiznet :**

**Source code with Explanation:**

#include <SPI.h> // Include the SPI library for communication with other devices.

#include <Ethernet.h> // Include the Ethernet library for network communication.

#include <PubSubClient.h> // Include the MQTT library for handling MQTT communication.

#include "HardwareSerial.h" // Include the hardware serial library for serial communication.

// Define the MAC address of the Ethernet shield.

byte mac[] = { 0xDE, 0xED, 0xBA, 35, 0xFE, 0xED };

// Define the local IP address for the Arduino.

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

// Define the IP address of the MQTT server or broker.

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

// Define the MQTT topic to which the Arduino will subscribe.

const char\* rgb\_values\_topic = "rgb\_values\_topic"; // Replace with the actual topic

// Define the pins for each color channel of the RGB LED.

const int redPin = A0;

const int greenPin = D14;

const int bluePin = D15;

// Create instances of the EthernetClient and PubSubClient classes.

EthernetClient ethClient;

PubSubClient client(ethClient);

// Initialize variables to store RGB color values.

int redValue = 0;

int greenValue = 0;

int blueValue = 0;

// Callback function executed when an MQTT message is received.

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

String message = "";

// Convert the payload bytes to a string.

for (int i = 0; i < length; i++) {

message += (char)payload[i];

}

// Find the positions of commas in the received message.

int commaIndex1 = message.indexOf(',');

int commaIndex2 = message.indexOf(',', commaIndex1 + 1);

// Check if the message is in the expected format "R,G,B".

if (commaIndex1 != -1 && commaIndex2 != -1) {

// Extract and convert RGB values from the message.

redValue = message.substring(0, commaIndex1).toInt();

greenValue = message.substring(commaIndex1 + 1, commaIndex2).toInt();

blueValue = message.substring(commaIndex2 + 1).toInt();

// Print the received RGB values to the serial monitor.

Serial.print("RGB: ");

Serial.print(redValue);

Serial.print(greenValue);

Serial.println(blueValue);

}

}

// Function to establish MQTT connection and subscribe to topics.

void reconnect() {

// Attempt to reconnect to the MQTT broker.

while (!client.connected()) {

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

// Connect to the broker with a client ID.

if (client.connect("arduinoClient35")) {

Serial.println("connected");

// Publish an announcement upon successful connection.

client.publish("rgb\_topic", "Values Sent!!");

// Subscribe to the designated topic for RGB values.

client.subscribe(rgb\_values\_topic);

} else {

// Print connection failure and retry after a delay.

Serial.print("failed, rc=");

Serial.print(client.state());

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

delay(5000);

}

}

}

// Setup function, executed once at the beginning.

void setup() {

// Configure the hardware serial port for communication.

Serial3.setRx(PC11);

Serial3.setTx(PC10);

Serial3.begin(9600);

// Initialize serial communication for debugging.

Serial.begin(9600);

// Configure the MQTT client with server and callback.

client.setServer(server, 1883);

client.setCallback(callback);

// Initialize Ethernet communication and allow time for setup.

Ethernet.begin(mac);

delay(1500);

// Set color channel pins as outputs.

pinMode(redPin, OUTPUT);

pinMode(greenPin, OUTPUT);

pinMode(bluePin, OUTPUT);

}

// Main loop function, executed repeatedly.

void loop() {

// Reconnect to MQTT broker if disconnected.

if (!client.connected()) {

reconnect();

}

// Maintain MQTT communication.

client.loop();

// Set RGB LED color based on received values.

analogWrite(redPin, 255 - redValue);

analogWrite(greenPin, 255 - greenValue);

analogWrite(bluePin, 255 - blueValue);

// Add any additional continuous operations here.

}

**Explanation:**

1. **callback Function:** The **callback** function deciphers incoming MQTT messages, converting bytes into a human-readable format. Triggered upon MQTT data arrival, it interprets "R,G,B" formatted messages containing RGB color info. By locating commas, it splits the data into individual color components. If the message structure fits, it extracts and converts values (like "255,0,0" to red 255, green 0, blue 0), showcasing them via serial print.
2. **reconnect Function:** **reconnect** persists as a negotiator between Arduino and MQTT. If the link disconnects, this function steps in, continuously attempting reconnection. Using a dedicated ID, it reconnects, signaling success via MQTT, and subscribing to the RGB topic. On failure, it reports, displays the client's state, and retries after 5 seconds.
3. **setup Function:** **setup** preps the stage before showtime. It arranges communication interfaces, initializes networking (Ethernet and MQTT), and pinpoints RGB LED control pins. Executed once at program start, it readies the environment for core operations.
4. **loop Function:** **loop** conducts the performance, orchestrating ongoing activities. It watches MQTT connection status, engaging the **reconnect** if lost. This loop continuously checks for incoming MQTT messages, enabling **callback** interpretation. MQTT communication sustains via **client.loop()**, while the function also manages RGB LED updates based on received values. The loop welcomes seamless integration of various tasks, supporting Arduino's multitasking capabilities.

The overall purpose of this code is to control an RGB LED's color remotely via MQTT messages. It listens for messages on the "rgb\_values\_topic," extracts RGB values from the received message, and sets the corresponding colors on the RGB LED. The Arduino will keep running this loop to maintain the MQTT connection and update the RGB LED color based on incoming messages.