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

**TABLE OF CONTENTS:**

1. **INTRODUCTION**
2. **HARDWARE USED**
3. **SOFTWARE USED**
4. **PROGRAMMING LANGUAGES**
5. **APPS AND THE ONLINE SERVICES USED**
6. **WORKFLOW**
7. **SETTING UP WITH PYTHON**
8. **SAMPLE UTTERENCES**
9. **OUTPUT**
10. **CONCLUSION**
11. **INTRODUCTION:**

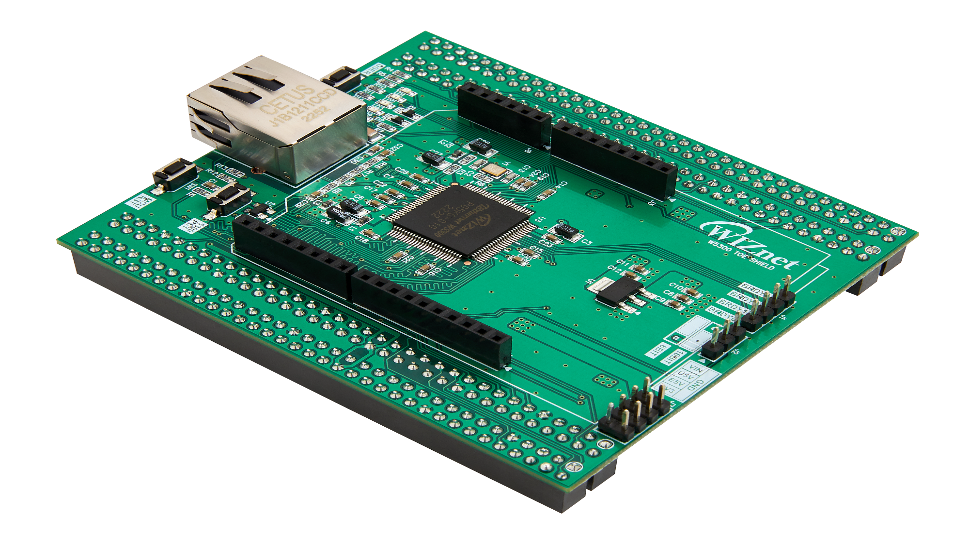
Welcome to the future of lighting innovation – the smart Lighting System! With its cutting-edge technology and seamless integration of voice commands, our system brings your dreams to life with a mere utterance. Imagine a world Where your words can paint your surrounding with vibrant hues and mesmerizing ambiance. With our Smart lighting System, you become the conductor, orchestrating a symphony of colours to suit your mood and preferences. Where Your voices becomes the pallete , and the lights are your canvas.

Introducing “Smart mood lighting system”. This cutting-edge system integrates voice commands and AI -driven text analysis to create a truly personalized and immersive experience. Basically with the Python programming we will give voice commands Which then communicates with the ChatGPT to accurately interpret then on basis of the interpretation it will extract the desired colour code and will take desired actions.

To implement this project, Wiznet-W5300 TOE SHIELD board which, is connected to the LED RGB, With the board we can give the commands to LED to give the particular ambiance of the room.

1. **HARDWARE USED:**

* WIZnet-W5300 TOE SHIELD + STM32-F429ZI board



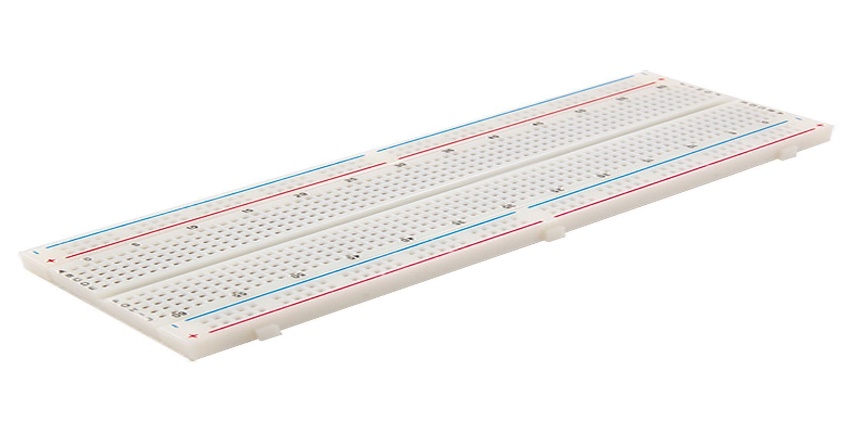
* LED RGB



* JUMPER WIRES



* BREAD BOARD



1. **SOFTWARES AND SERVICES USED**

* MQTT SERVICE
* ARDUINO IDE
* PYCHARM

1. **PROGRAMMING LANGUAGES**

* C++
* PYTHON

1. **APPS AND THE ONLINE SERVICES USED:**
2. **MQTT-BROKER**
3. **OPENAI**
4. **Google's speech-to-text API**

**MQTT-BROKER:**

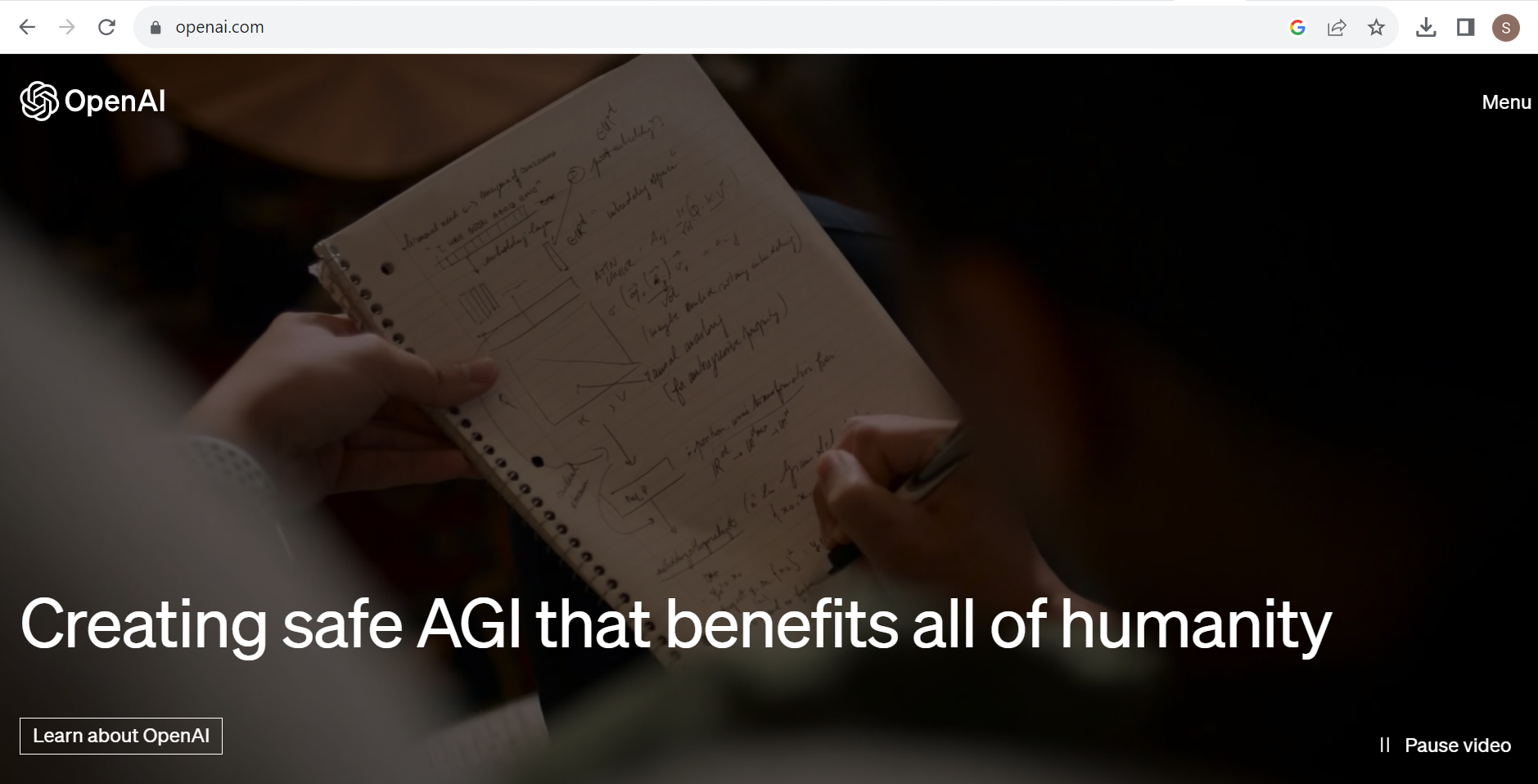
MQTT acts as a broker, facilitating communication between Python and Arduino. Python sends RGB values via MQTT to the Arduino. Arduino, configured to receive MQTT messages, interprets the RGB data and controls the connected RGB LED accordingly. This enables seamless remote control of the RGB LED’s colour using MQTT protocol, bridging Python and Arduino for effective IoT applications.

**OPEN AI :**

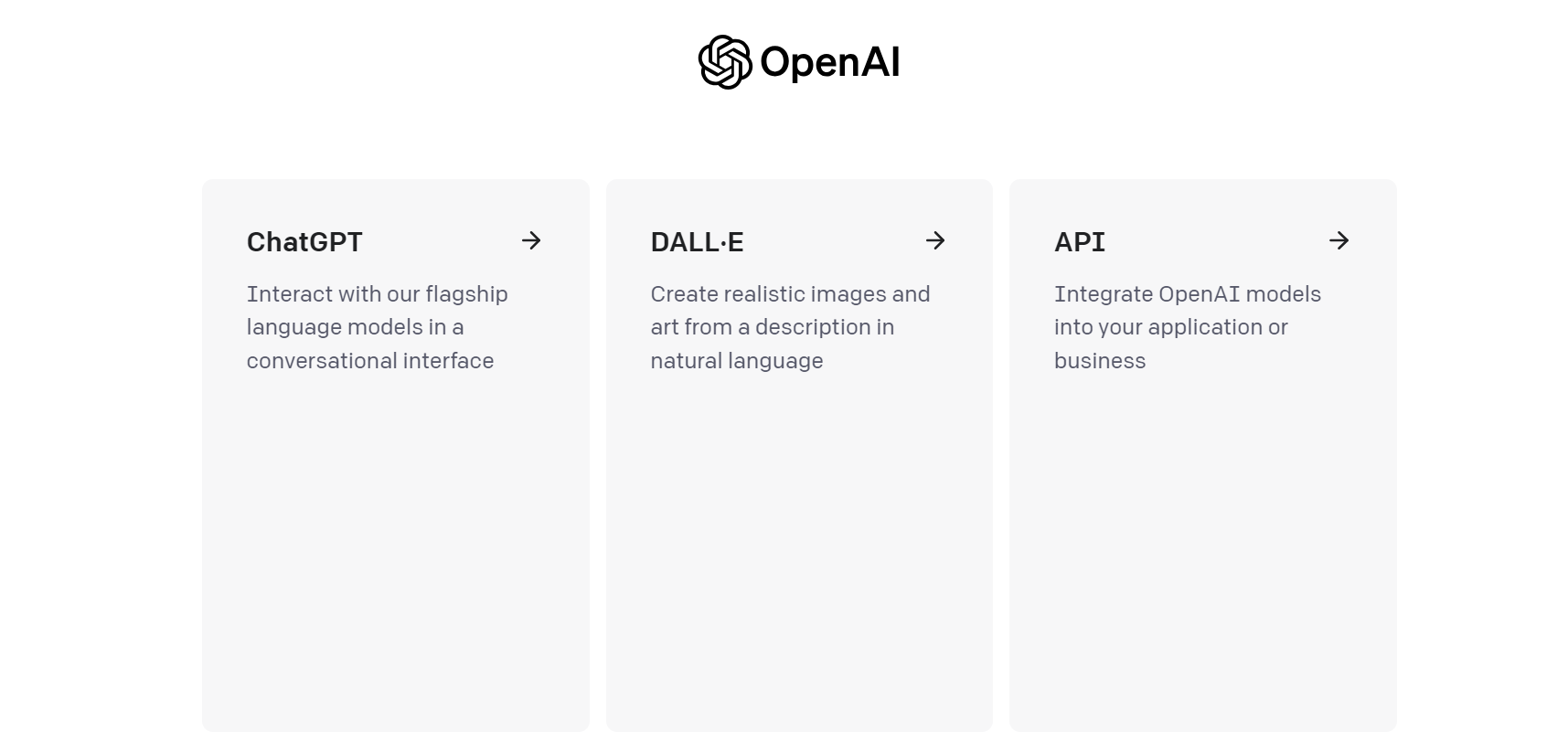
OpenAI provides web services that allow developers to access and utilize its advanced natural language processing models, like the GPT-3. These web services enable applications to integrate powerful language capabilities, such as text generation, language translation, sentiment analysis, and more. Developers can make API calls to interact with these models over the internet, enabling the integration of OpenAI's technology into a wide range of software applications and services.

Setting up with OpenAi :

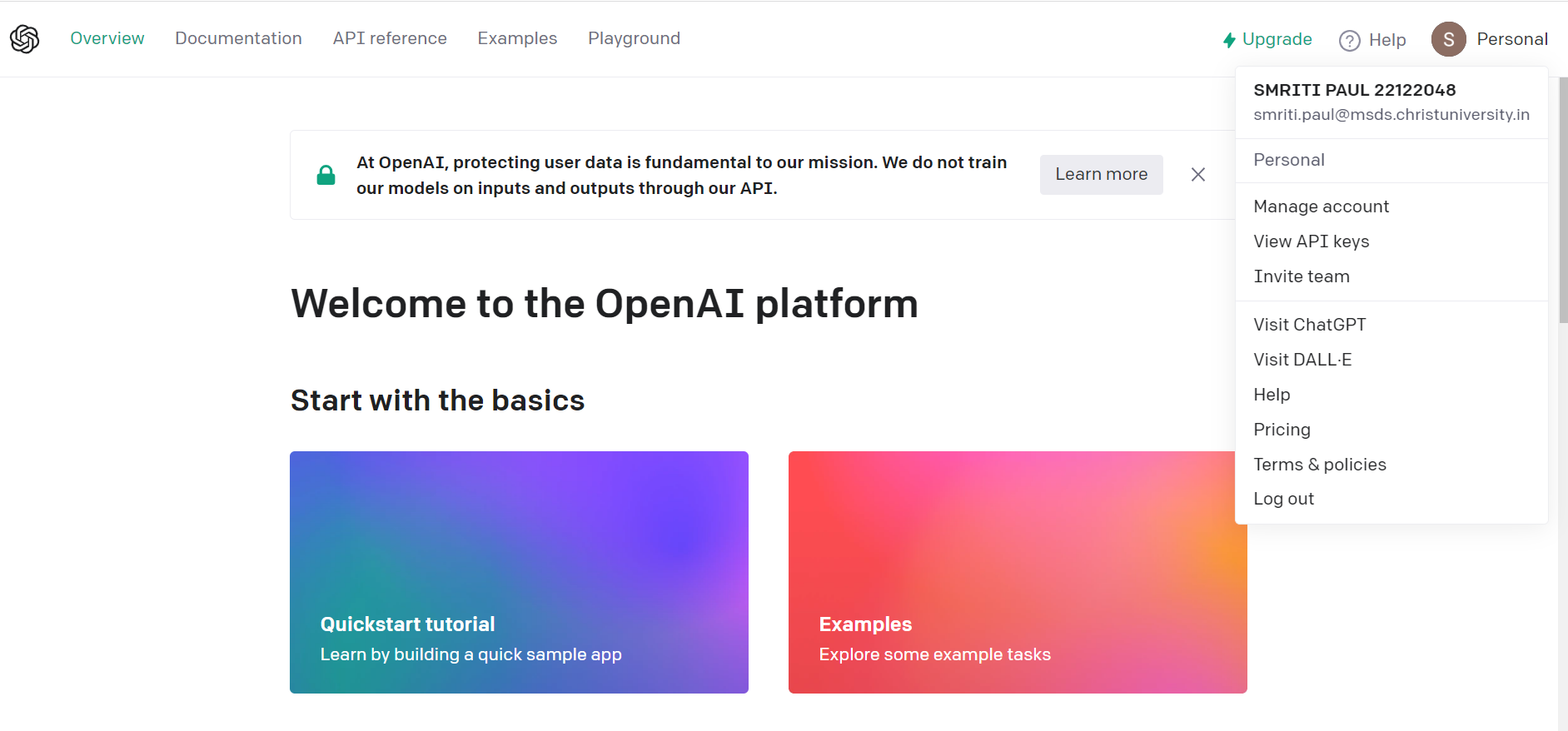
1. <https://openai.com/> Go to this link



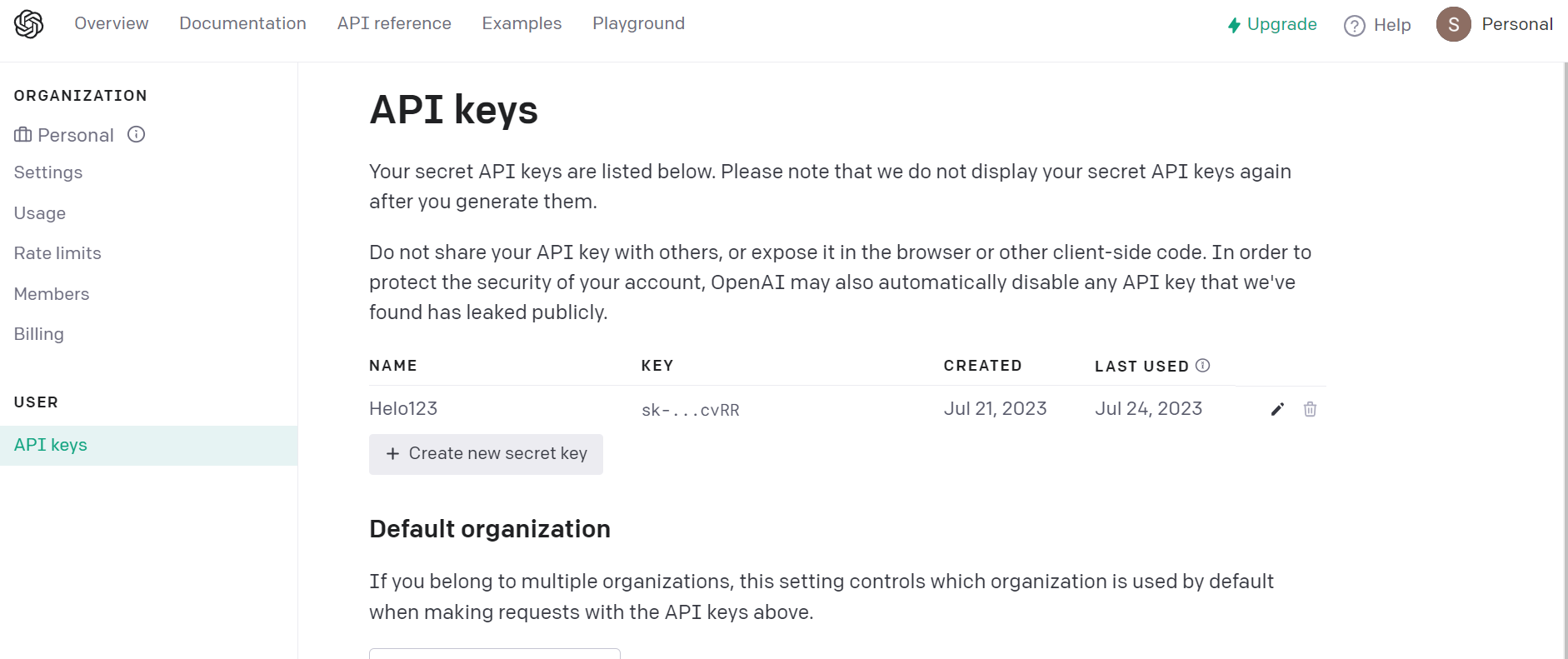
1. Go to menu and then Sign up (Create a new account) or otherwise just Log in

****

1. Then you can see above three options go to API for generating the API Key.



1. Then go to 🡪 Personal 🡪 view API Keys



From Create new secret key option you can create a new API , It will generate once and have take a copy of that.

**GOOGLE TEXT TO SPEECH:**

recognized\_text = recognizer.recognize\_google(audio)

this code snippet utilizes the Google text-to-speech API service through the **recognize\_google** function provided by the SpeechRecognition library in Python. This function sends the recorded audio data to Google's servers for speech recognition processing, and then it returns the recognized text back to your Python program.In this case, the **recognize\_google** function is using the Google Web Speech API to perform the speech-to-text conversion, allowing you to transcribe spoken language into written text.

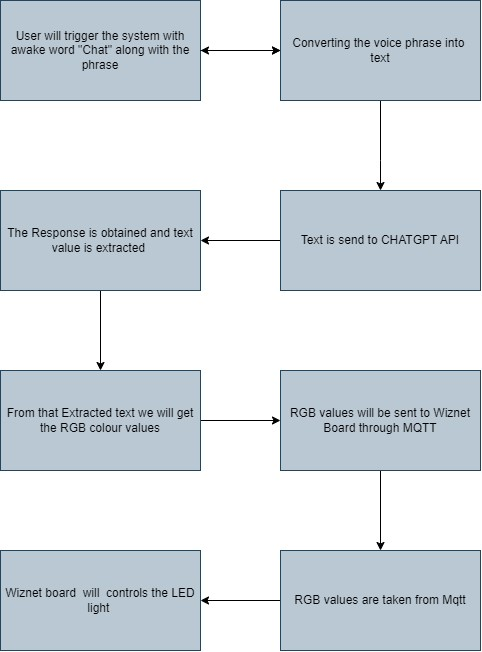
1. **WORKFLOW**

The workflow of this project “Smart Mood light” for turning on different colour on the basis of your desires command where you just gives the names of the shades you want on the basis of your mood and sometimes you just need explain the view and it will interpret the colour from that pharse and turn on the coloured LED, All the you don’t need say turn on for turning on the light.

1. Voice Input: The Systems Starts by receiving voice input from the user. User can provide commands like “I want cherry blossoms view” or simply describe a colour or view they desire. Also Need add the Special Word “Chat” to make the command recognise.
2. Speech -to -text Conversion: The voice input is converted into text using a Speech Recognition library in Python.So This process ensures that system can work with textual data for further analysis.
3. Text analysis: Once the voice command is converted into text, the System will go to ChatGPT inside this where the text analysis will happen through- GPT-3.5 model And give the desire command. This components identifies keywords, colours, and phrases related to views or moods mentioned by the user.
4. Colour interpretation: Based on the analyzed text, the system interprets the desired colour or view. For instance, if the user says “Cherry blossoms view”, the system recognizes the associated pink colour.
5. Colour Mapping: The interpreted colour is then mapped to specific RGB values or colour codes used by the LED lighting system. This mapping ensures that the system can accurately represent the desired colours using the available LEDs.
6. LED Control: The system sends the mapped colour information to the LED control module (WIZnet board), which then activates the corresponding LEDs to emit the desired colour. There is no need for user to explicitly say “turn on the light” since the system automatically interprets the commands.
7. Feedback: The system provides feedback to the user, confirming the successful interpretation and activation of the desired color. This feedback can be in the form of a voice response or visual indicators.
8. Continuous Listening: The Smart Lighting System continuously listens for new voice commands, allowing users to change colors or views on-the-fly by providing new input.

By combining voice recognition, text analysis, colour interpretation, and LED control, the smart Lighting System ensures a seamless and personalized experience, making it a truly innovative and intuitive way to interact with lighting in your living space.

**FLOW CHART:**



**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

**Code:**

import openai

# Set up OpenAI API key  
openai.api\_key = "sk-b7\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*oPb4L60"

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()

This is the code for using Open Ai with python with the API key I will set up with python. Then

**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 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.

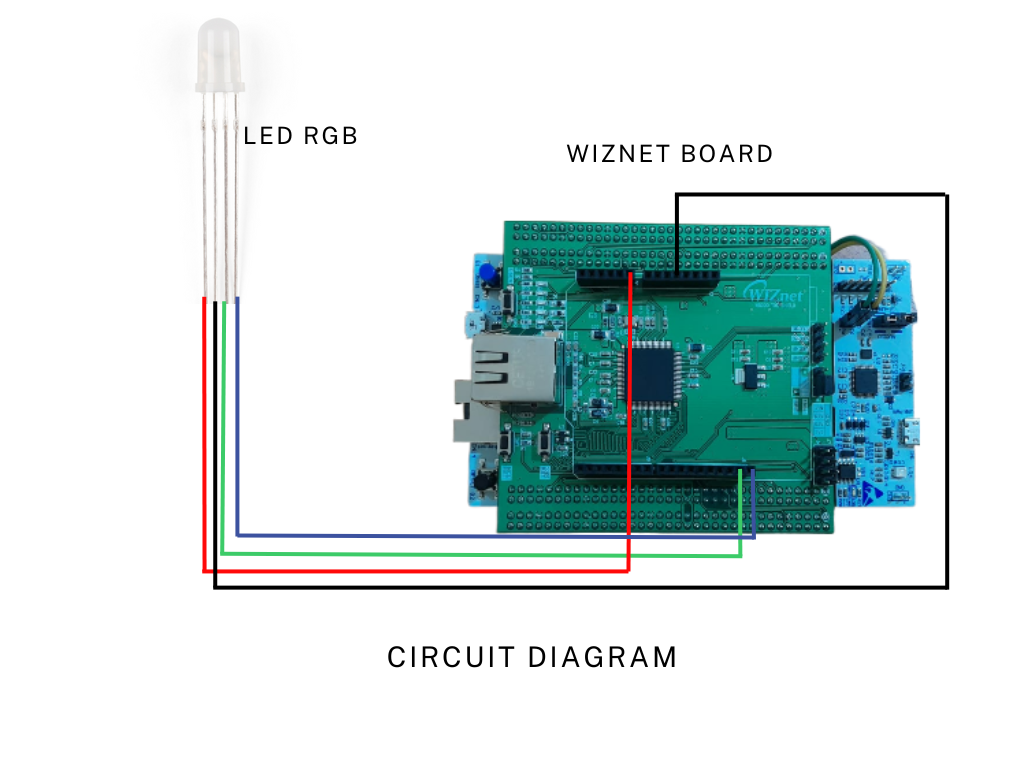
Installation: To install the pyttsx3 library, you can use pip:

pip install pyttsx3

Another two libraries are used here threading, re. “threading” -- Threads are particularly useful when you want to perform tasks simultaneously without blocking the main program's execution. This can improve the responsiveness of your application, especially for I/O-bound tasks.

“re” -- This library is part of Python's standard library and provides support for regular expressions. Regular expressions are a powerful tool for pattern matching and text manipulation. The re library allows you to search, find, and replace specific patterns within strings.

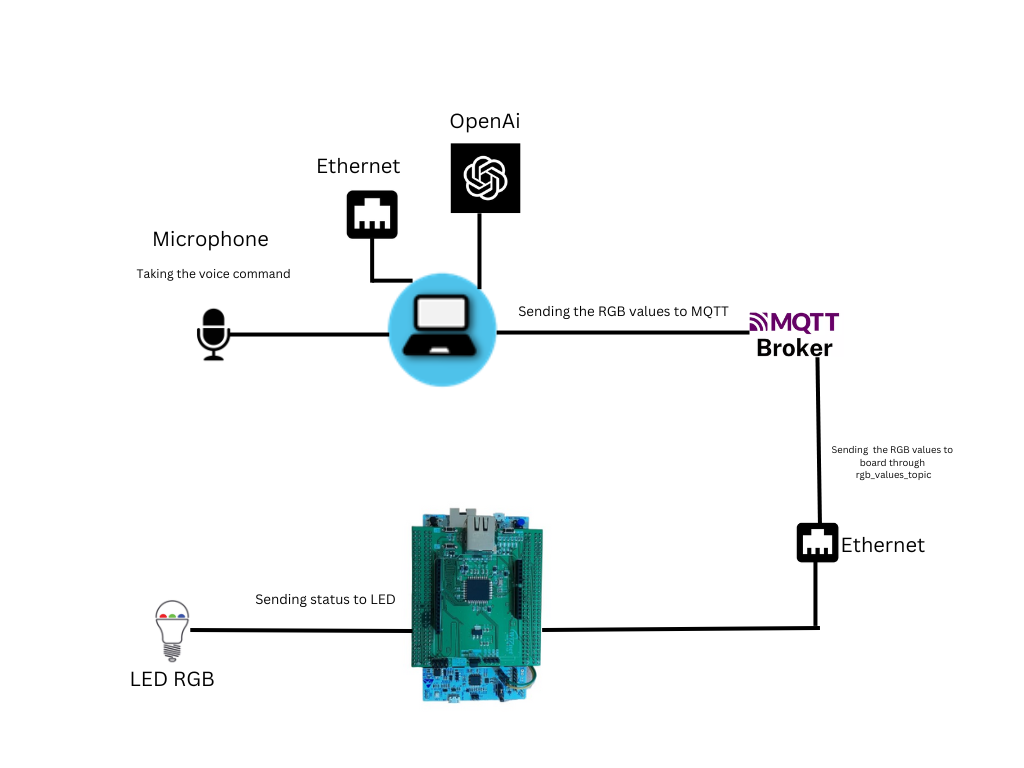
**CIRCUIT CONNECTION:**



LED RGB has 4 pins R (red pin), Common anode, G (green pin), B (blue pin) which is connected with W5100s-EVB-pico board

* Red pin is connected with A0
* VCC is connected with 5V
* Green pin is connected with D14
* Blue pin is connected with D15

**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:

Turning on:

* Chat Turn on the light\4’
* Chat I can’t see anything
* Chat I want to look
* Make it bright
* Chat Make me see
* Chat Turn on

Turning on (colour):

* Chat I want to see cherry blossom’s view

(Turn on the pink light)

* Chat Make it greenery

(Turn on the green light)

* Chat I want to see forest view

(Turn on light green light)

* Chat Please make it sunset like

(Turn on the orange)

* Chat can I have purple sky like view

(Turn on the purple light)

* Chat Turn on magenta

(Turn on magenta light)

* Chat Let’s have a party

(Turn on all hue colours)

Turning off:

* Chat I want darkness
* Chat Turn off
* Chat I want to sleep
* Chat I want it to be dark
* Chat Let there be darkness

**EXAMPLES:**

|  |  |
| --- | --- |
| You said: Chat turn On Magenta  AI Response: Turn on magenta light |  |
| You said: Chat Can I have cherry blossom’s view  AI Response: Turn on pink light |  |
|  |  |
| You said: Chat can I see midnight view  AI Response: Turn on light midnight blue |  |
|  |  |
| You said: Chat Make it lime green  AI Response: Turn on light lime green |  |
| You said: Chat can I have a sunset view  AI Response: Turn on the orange light |  |

**OUTPUT:**

[](https://www.youtube.com/embed/zpPhmPdvTZE?feature=oembed)

Video Link : <https://youtu.be/zpPhmPdvTZE>

**CONCLUSIONS:**

A smart mood light with ChatGPT + voice recognition + TTS is a device that can change its color and brightness in response to a user's voice commands or the content of a conversation. The ChatGPT component allows the light to generate text-based responses to user queries, while the voice recognition and TTS components allow it to understand and respond to spoken commands. This type of light has the potential to be a valuable tool for people with disabilities, as it can provide them with a way to control their environment without having to use their hands. It could also be used to create a more immersive and interactive experience for users of smart home devices.Overall, the combination of ChatGPT, voice recognition, and TTS has the potential to make smart mood lights more versatile and user-friendly. This could lead to increased adoption of these devices by people with disabilities and other users who find traditional smart home controls difficult to use.

**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 <Ethernet.h>

#include <PubSubClient.h>

// Update these with values suitable for your network.

byte mac[] = { 0xDE, 0xED, 0xBA, 35, 0xFE, 0xED }; // MAC address of the Ethernet shield

IPAddress ip(172, 16, 0, 100); // Static IP address for the Arduino

IPAddress server(44, 195, 202, 69); // IP address of the MQTT broker/server

const char\* rgb\_values\_topic = "rgb\_values\_topic"; // MQTT topic to subscribe to (replace with actual topic name)

// Define the pins for each color channel

const int redPin = 9; // Pin number for the red channel of the RGB LED

const int greenPin = 5; // Pin number for the green channel of the RGB LED

const int bluePin = 6; // Pin number for the blue channel of the RGB LED

EthernetClient ethClient; // Create an instance of the EthernetClient class for Ethernet communication

PubSubClient client(ethClient); // Create an instance of the PubSubClient class for MQTT communication

int redValue = 0; // Variable to store the intensity value for the red channel (0-255)

int greenValue = 0; // Variable to store the intensity value for the green channel (0-255)

int blueValue = 0; // Variable to store the intensity value for the blue channel (0-255)

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

// This function is called when an MQTT message is received.

// Extract the message payload and update the RGB values accordingly.

// The payload is expected to be in the format "R,G,B" (e.g., "255,0,0" for red).

String message = ""; // Create an empty string to hold the payload content

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

message += (char)payload[i]; // Convert the payload bytes to a string

}

// Extract the individual RGB values from the payload string

int commaIndex1 = message.indexOf(','); // Find the index of the first comma

int commaIndex2 = message.indexOf(',', commaIndex1 + 1); // Find the index of the second comma

// If the payload format is correct (contains two commas),

// extract and convert the RGB values from the payload string

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

redValue = message.substring(0, commaIndex1).toInt(); // Extract and convert the red value

greenValue = message.substring(commaIndex1 + 1, commaIndex2).toInt(); // Extract and convert the green value

blueValue = message.substring(commaIndex2 + 1).toInt(); // Extract and convert the blue value

}

}

void reconnect() {

// This function is called to reconnect to the MQTT broker if the connection is lost.

while (!client.connected()) {

Serial.print("Attempting MQTT connection..."); // Print message indicating the connection attempt

// Attempt to connect to the MQTT broker with a client ID of "arduinoClient35"

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

Serial.println("connected"); // Connection successful

client.publish("rgb\_topic", "Values Sent!!"); // Publish a message to the topic "rgb\_topic"

client.subscribe(rgb\_values\_topic); // Subscribe to the topic specified by "rgb\_values\_topic"

} else {

Serial.print("failed, rc="); // Connection failed, print the failure reason code

Serial.print(client.state()); // Print the state of the PubSubClient

Serial.println(" try again in 5 seconds"); // Print a message indicating the next retry

delay(5000); // Wait 5 seconds before attempting to reconnect

}

}

}

void setup()

{

Serial.begin(57600); // Initialize the serial communication for debugging

Ethernet.init(17); // Initialize the Ethernet shield with the CS (Chip Select) pin number

client.setServer(server, 1883); // Set the MQTT server and port for the PubSubClient

client.setCallback(callback); // Set the callback function to handle incoming MQTT messages

Ethernet.begin(mac); // Initialize the Ethernet connection with the specified MAC address

delay(1500); // Wait for the Ethernet hardware to initialize

pinMode(redPin, OUTPUT); // Set the redPin as OUTPUT

pinMode(greenPin, OUTPUT); // Set the greenPin as OUTPUT

pinMode(bluePin, OUTPUT); // Set the bluePin as OUTPUT

}

void loop()

{

if (!client.connected()) {

reconnect(); // If not connected to the MQTT broker, attempt to reconnect

}

client.loop(); // Process MQTT messages and maintain the connection

// Set the color of the RGB LED based on the received RGB values

// The "analogWrite" function sets the intensity of each color channel (0-255)

analogWrite(redPin, 255 - redValue); // Subtract the intensity value from 255 since the LED is common anode

analogWrite(greenPin, 255 - greenValue); // Subtract the intensity value from 255 since the LED is common anode

analogWrite(bluePin, 255 - blueValue); // Subtract the intensity value from 255 since the LED is common anode

// Add any other code here that you want to run continuously

}

**Explanation:**

* + void callback(char\* topic, byte\* payload, unsigned int length): This function is the MQTT callback, which gets called whenever a message is received on a subscribed topic. It extracts the RGB values from the received payload and updates the redValue, greenValue, and blueValue variables accordingly.
  + void reconnect(): This function handles the reconnection to the MQTT broker in case the connection is lost. It attempts to reconnect to the broker until the connection is successful. If the connection is established, it publishes an announcement message to the "rgb\_topic" and subscribes to the "rgb\_values\_topic."
  + void setup(): This is the setup function that runs once when the Arduino is powered on or reset. It initializes the serial communication, sets up the Ethernet connection, and configures the pins connected to the RGB LED (red, green, and blue) as OUTPUT.
  + void loop(): This is the main loop function that runs continuously after the setup. It first checks if the MQTT client is connected. If not, it calls the reconnect() function to establish the MQTT connection. It then calls the MQTT client's loop() function to maintain the MQTT connection and handle incoming messages. After that, it sets the color of the RGB LED based on the values received through MQTT, using analogWrite to adjust the intensity of each color.

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.