**[AIoT]Smart Clock System+ Voice-Recognition + TTS**

**TABLE OF CONTENTS:**

1. **INTRODUCTION**
2. **HARDWARE USED**
3. **SOFTWARE USED**
4. **PROGRAMMING LANGUAGES**
5. **WORKFLOW**
6. **SETTING UP WITH PYTHON**
7. **OUTPUT**
8. **CONCLUSION**
9. **INTRODUCTION:**

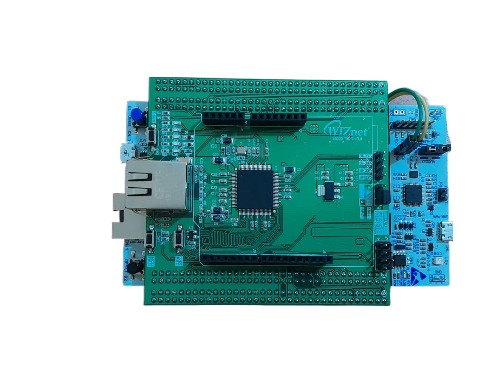
The Smart Clock project is a remarkable embodiment of modern technology, seamlessly integrating voice recognition with a range of indispensable functions to enhance everyday life. This innovative clock not only displays the current time but offers an array of versatile features that elevate its utility. With its vibrant OLED screen, powered by the advanced Wiznet pico connection, the clock provides an intuitive visual interface, ensuring a user-friendly experience.

The clock's voice recognition capabilities make it a truly hands-free device, allowing users to effortlessly set timers, activate the stopwatch, access world clock information, set alarms, and create reminders. Imagine effortlessly starting the stopwatch with a simple command, and watching the precise timing displayed on the crisp OLED screen. Need to stay organized? The clock's reminder function will keep you on track, displaying important alerts right on the screen.

Whether you're a multitasker in need of precise time management or someone who values convenience and efficiency, this Smart Clock is designed for you. Its seamless integration of voice commands and visual display ensures that accessing its functions is a breeze. The clock doesn't just tell time; it becomes a reliable companion, helping you navigate your daily tasks effortlessly while adding a touch of elegance to your living space. Experience the future of timekeeping with our innovative Smart Clock

1. **HARDWARE USED:**

* WIZnet-W5300 TOE SHIELD + STM32-F429ZI board



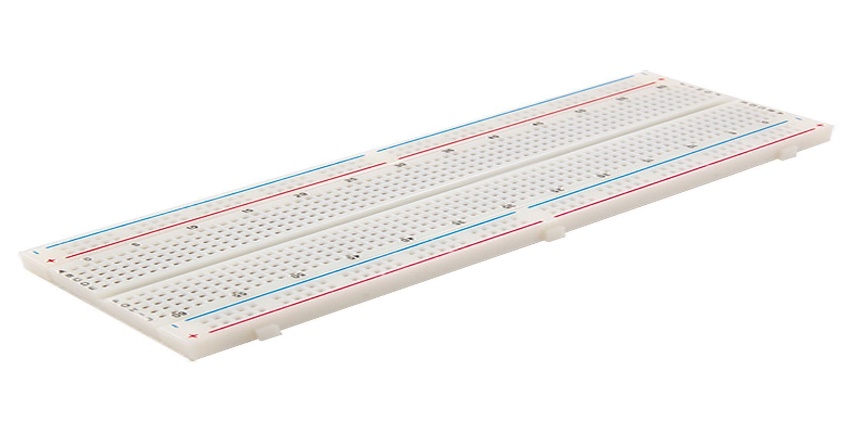
* OLED SCREEN

****

* JUMPER WIRES



* BREAD BOARD



* USB MICROPHONE

****

1. **SOFTWARES AND SERVICES USED**

* MQTT SERVICE
* ARDUINO IDE
* PYCHARM

1. **PROGRAMMING LANGUAGES**

* C++
* PYTHON

**MQTT-BROKER:**

MQTT acts as a broker, facilitating communication between Python and Arduino. Python sends values via MQTT to the Arduino. Arduino, configured to receive MQTT messages, and print them to the connected OLED SCREEN accordingly. This enables seamless remote control of the OLED SCREEN using MQTT protocol, bridging Python and Arduino for effective IoT applications.

**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 Smart Clock project has a well-defined workflow that begins with a welcome message and operates through a series of voice commands, with the wake-up word "clock." Here's a breakdown of the input function and several key functions:

1. \*\*Input Function\*\*:

- Upon program start, the Smart Clock welcomes the user with a friendly message, setting the stage for interaction.

- The user's input is primarily in the form of voice commands, initiated by saying the wake-up word "clock" followed by specific commands.

2. \*\*Set Timer\*\*:

- Command: "clock set timer."

- Function: The user can set a timer for a specific duration(in seconds) using this command.

- Interaction: The Smart Clock responds by asking the user to specify the time for the timer, and the clock starts counting down.

3. \*\*Set Stopwatch\*\*:

- Command: "clock set stopwatch."

- Function: The user can initiate a stopwatch using this command, there are three command start, stop,exit.

- Interaction: The Smart Clock acknowledges the command and starts the stopwatch, displaying the time on the OLED screen.

4. \*\*Set Alarm\*\*:

- Command: "clock set alarm."

- Function: The user can set an alarm for a particular time using this command.

- Interaction: The Smart Clock prompts the user to specify the alarm time, and when the set time is reached, the alarm activates with a notification on the screen.

5. \*\*Set Reminder\*\*:

- Command: "clock set reminder."

- Function: The user can create a reminder for a specific event or task.

- Interaction: The Smart Clock prompts the user to input the reminder details, and at the specified time, it displays the reminder on the screen.

6. \*\*Start World Clock\*\*:

- Command: "clock start world clock."

- Function: The user can access the world clock feature to check the time in different time zones.

- Interaction: The Smart Clock displays the current time for various pre-configured world clock locations on the OLED screen.

7. \*\*Start Current Time\*\*:

- Command: "clock start current time."

- Function: The user can request the current local time.

- Interaction: The Smart Clock displays the current local time on the OLED screen.

8. \*\*Turn Off/Exit\*\*:

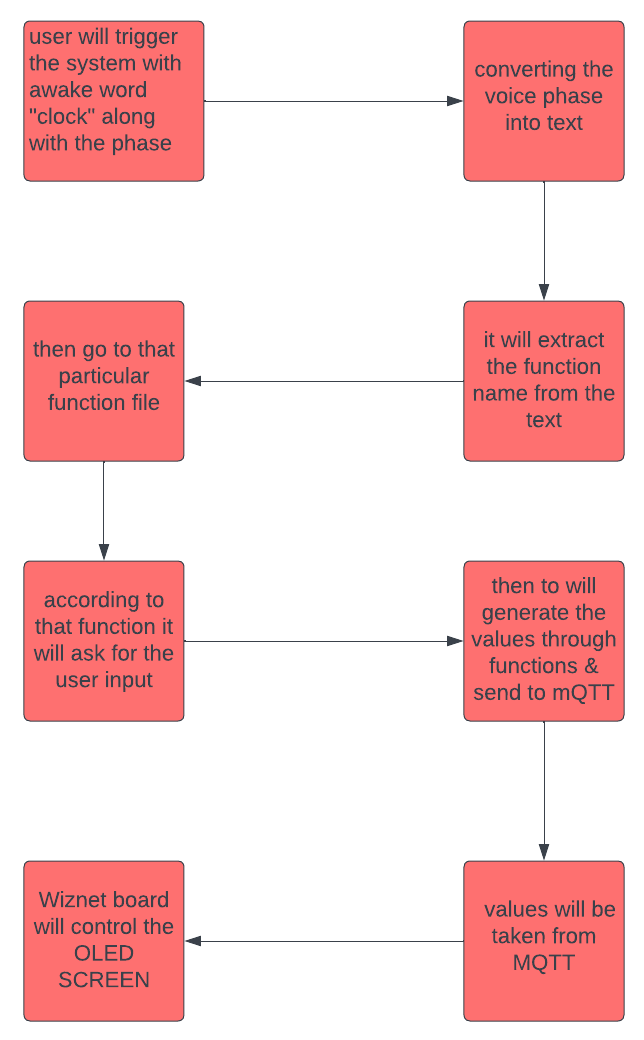
- Command: "turn off."

- Function: The user can exit the program and turn off the Smart Clock.

- Interaction: The Smart Clock acknowledges the command and gracefully shuts down the program.

The combination of voice recognition, the OLED screen, and these functions creates a dynamic and user-friendly experience, making the Smart Clock an essential companion for time management and daily organization.

**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 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.

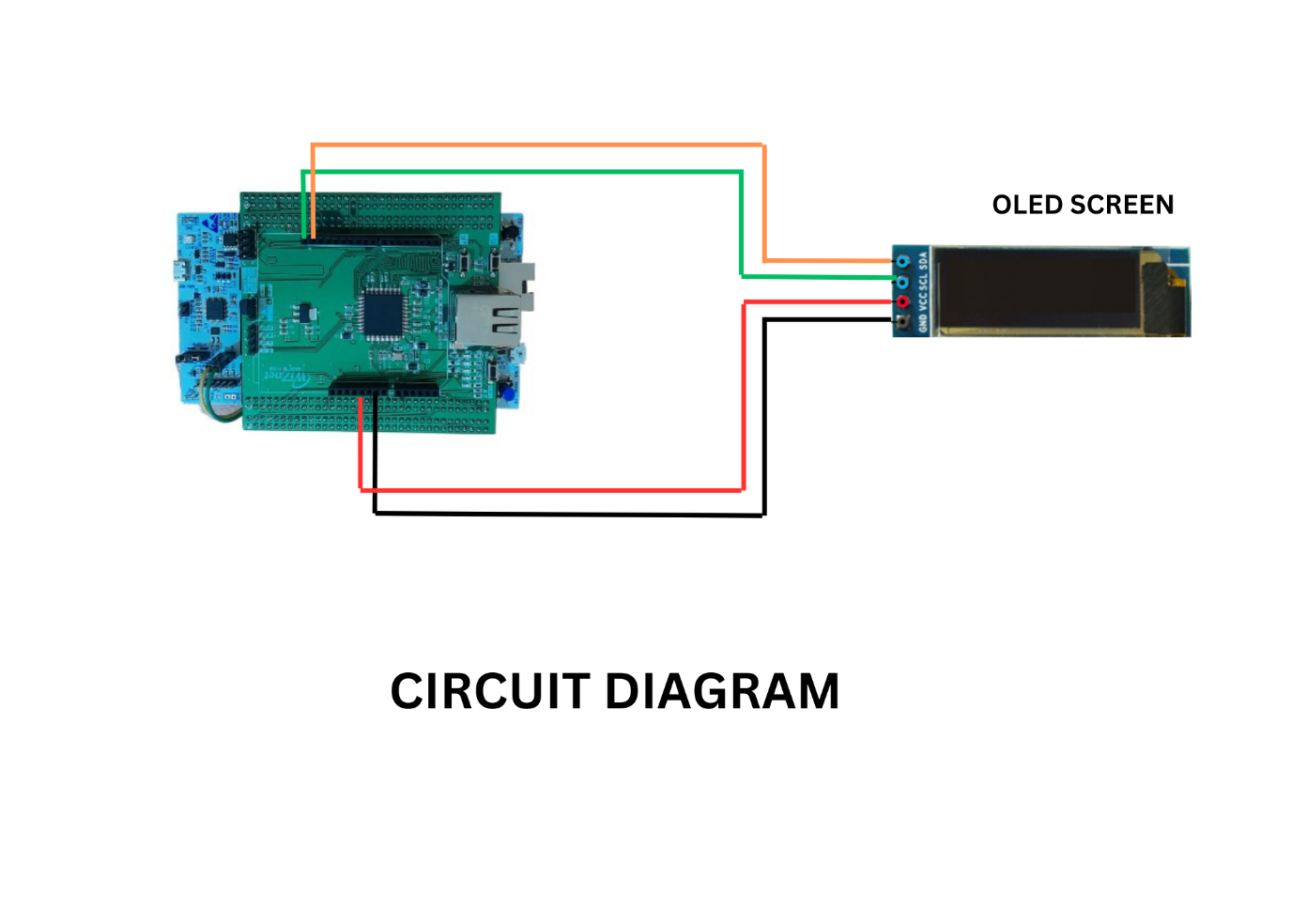
Installation: To install the pyttsx3 library, you can use pip:

pip install pyttsx3

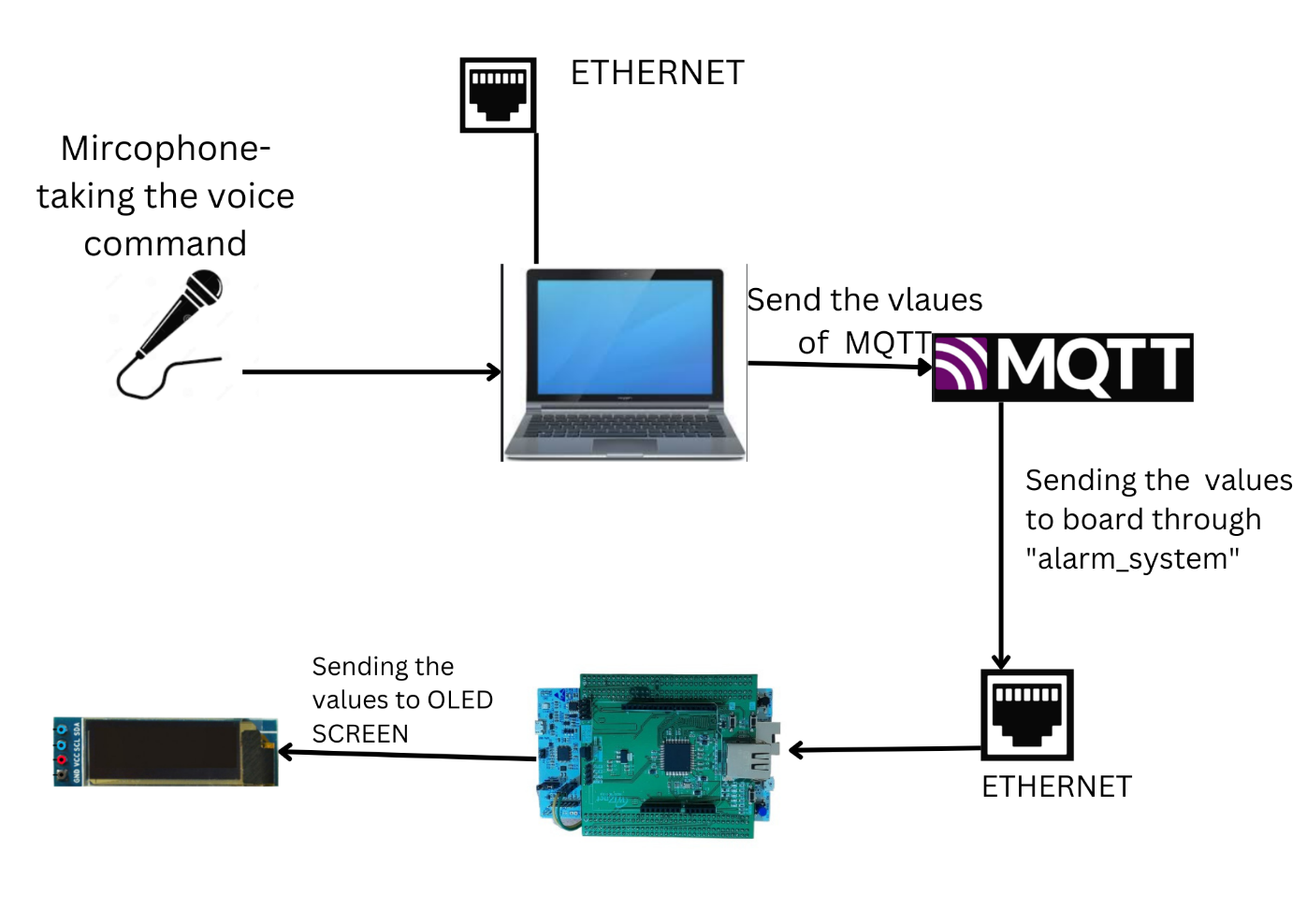
Another two libraries are used here threading, re. “threading” -- Threads are particularly useful when you want to perform tasks simultaneously without blocking the main program's execution. This can improve the responsiveness of your application, especially for I/O-bound tasks.

“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:**



**CONNECTION DIAGRAM:**



**CONCLUSIONS:**

**Certainly! The Smart Clock System project leverages various technologies to create a versatile and interactive time management solution. By integrating speech recognition, text-to-speech conversion, and MQTT communication, the system enables users to control and interact with clock-related functionalities using their voice. Users can set alarms, timers, reminders, and inquire about the current time for different timezones and countries. The project's architecture empowers users to conveniently interact with the clock functionalities through natural language, enhancing accessibility and usability. The MQTT broker facilitates real-time communication and information sharing between different modules of the system, enabling seamless integration. The project not only offers practical time-related utilities but also serves as a showcase of how emerging technologies can be integrated to create innovative and user-friendly solutions for everyday tasks.**

**CODE EXPLAINITION:**

**Python Explanation:**

**Main.py**

**import pyttsx3  
import speech\_recognition as sr  
import importlib  
  
# Dictionary of recognized commands and their corresponding module names  
COMMANDS = {  
 "set timer": "timer",  
 "start current time": "current\_time",  
 "set stopwatch": "stopwatch",  
 "set reminder": "reminder",  
 "set alarm": "alarm",  
 "start world clock": "world\_clock",  
 "turn off": "exit" # Changed from "turn off"  
}  
  
# Function to convert text to speech  
def speak(text):  
 engine = pyttsx3.init()  
 engine.say(text)  
 engine.runAndWait()  
  
# Function to get voice input from the user  
def get\_voice\_input():  
 recognizer = sr.Recognizer()  
 with sr.Microphone() as source:  
 recognizer.adjust\_for\_ambient\_noise(source, duration=1)  
 audio = recognizer.listen(source)  
 try:  
 return recognizer.recognize\_google(audio).lower()  
 except sr.UnknownValueError:  
 return ""  
 except sr.RequestError:  
 speak("Sorry, I am unable to access the speech recognition service. Please try again later.")  
 return ""  
  
# Function to extract the command from user input  
def extract\_command(user\_input):  
 for command in COMMANDS:  
 if user\_input.startswith(f"clock {command}"):  
 return command  
 return None  
  
# Function to execute the appropriate module based on the command  
def execute\_module(command):  
 if command == "exit":  
 speak("Turning off the system. Goodbye!")  
 return False  
 elif command in COMMANDS:  
 module\_name = COMMANDS[command]  
 try:  
 module = importlib.import\_module(module\_name)  
 module.main() # Execute the module once  
 del module # Remove the reference to the module  
 except ImportError:  
 speak("Command module not found. Please try again.")  
 else:  
 speak("Command not recognized. Please try again.")  
 return True  
  
# Main function that listens for user input and executes commands  
def main():  
 speak("Welcome to the Smart Clock System. You can wake me up by saying 'clock', followed by your command.")  
 speak('"there are some command like set alarm,'  
 ' set reminder,'  
 ' set timer,'  
 ' set stopwatch,'  
 ' start current time,'  
 ' start world clock,'  
 ' turn off to close the smart clock system"')  
  
 while True:  
 user\_input = get\_voice\_input()  
  
 if "clock" in user\_input:  
 command = extract\_command(user\_input)  
 if command:  
 if not execute\_module(command):  
 break  
 elif "turn off" in user\_input:  
 speak("Turning off the clock system. Goodbye!")  
 break  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()**

**Explanation for main.py:**

**Certainly! This code represents a simple voice-controlled smart clock system using Python. Let's break down the different components and their functionalities:**

**1. \*\*Import Statements:\*\***

**- The code begins by importing necessary modules: `pyttsx3` for text-to-speech conversion, `speech\_recognition` (often abbreviated as `sr`) for speech recognition, and `importlib` for dynamic module importing.**

**2. \*\*COMMANDS Dictionary:\*\***

**- This dictionary maps recognized voice commands to corresponding module names. Each command is associated with a specific action that the smart clock can perform.**

**3. \*\*`speak` Function:\*\***

**- This function utilizes the `pyttsx3` module to convert text into speech. It initializes a speech synthesis engine, reads the input text, and runs the engine to produce speech.**

**4. \*\*`get\_voice\_input` Function:\*\***

**- This function handles the process of capturing audio input from the user's microphone using the `speech\_recognition` module.**

**- It uses noise adjustment to account for ambient noise and then listens for audio input.**

**- The recorded audio is passed to Google's speech recognition service to convert it into text.**

**- If successful, the recognized text is returned in lowercase; otherwise, an empty string is returned.**

**5. \*\*`extract\_command` Function:\*\***

**- This function extracts the command from the user input.**

**- It checks if the input starts with the phrase "clock" (e.g., "clock set timer"), and if so, it extracts the command from the input.**

**- The extracted command is then checked against the predefined commands in the `COMMANDS` dictionary.**

**6. \*\*`execute\_module` Function:\*\***

**- This function executes the appropriate module based on the recognized command.**

**- If the command is "exit," the system is turned off with a goodbye message.**

**- If the command is recognized and corresponds to a valid module, that module is imported and executed using the `main()` function defined in the module.**

**- If the command is not recognized or the module import fails, an appropriate response is given.**

**7. \*\*`main` Function:\*\***

**- The main function serves as the entry point of the program.**

**- It starts by welcoming the user and providing a list of available commands.**

**- It enters a loop where it continuously listens for user input using the microphone.**

**- If the input contains the phrase "clock," the system extracts the command and attempts to execute it using the `execute\_module` function.**

**- If the input contains "turn off," the system is shut down with a goodbye message.**

**8. \*\*`\_\_name\_\_` Check and Execution:\*\***

**- The script is designed to be run directly. The code block under `if \_\_name\_\_ == "\_\_main\_\_":` ensures that the `main()` function is executed when the script is run as the main program.**

**In summary, this script simulates a voice-controlled smart clock system that listens for commands, recognizes them using speech recognition, executes the corresponding actions using dynamically imported modules, and responds using text-to-speech conversion.**

**timer.py**

**import speech\_recognition as sr  
import pyttsx3  
import time  
import paho.mqtt.client as mqtt  
  
MQTT\_BROKER = "broker.emqx.io"  
MQTT\_PORT = 1883  
MQTT\_TOPIC = "alarm\_system"  
  
def speak(text):  
 *"""Converts text to speech."""* engine = pyttsx3.init()  
 engine.say(text)  
 engine.runAndWait()  
  
def get\_voice\_input():  
 *"""Captures voice input and returns the recognized text."""* recognizer = sr.Recognizer()  
  
 with sr.Microphone() as source:  
 recognizer.adjust\_for\_ambient\_noise(source, duration=1)  
 audio = recognizer.listen(source)  
  
 try:  
 return recognizer.recognize\_google(audio).lower()  
 except sr.UnknownValueError:  
 return ""  
 except sr.RequestError:  
 speak("Sorry, I am unable to access the speech recognition service. Please try again later.")  
 return ""  
  
def set\_timer():  
 *"""Allows the user to set a timer using voice input."""* speak("Please tell me the duration for the timer in seconds.")  
 timer\_duration = get\_voice\_input()  
  
 try:  
 timer\_duration = int(timer\_duration)  
 speak(f"Timer set for {timer\_duration} seconds.")  
  
 return timer\_duration  
 except ValueError:  
 speak("Invalid duration format. Please try again.")  
 return set\_timer()  
  
def countdown\_timer(duration):  
 *"""Displays the timer countdown on the console and sends the timer value to MQTT topic."""* mqtt\_client = mqtt.Client()  
 mqtt\_client.connect(MQTT\_BROKER, MQTT\_PORT)  
  
 for seconds in range(duration, -1, -1):  
 timer\_str = f"Timer- {seconds // 60:02d}:{seconds % 60:02d}"  
 print(timer\_str, end='\r') # Use '\r' to overwrite the previous line in the console  
 mqtt\_client.publish(MQTT\_TOPIC, timer\_str) # Publish timer value to MQTT topic  
 time.sleep(1)  
  
 print("Timer ended")  
 mqtt\_client.publish(MQTT\_TOPIC, "Timer ended") # Publish timer ended message to MQTT topic  
 mqtt\_client.disconnect()  
  
def main():  
 speak("Welcome to the Timer function using voice input.")  
 timer\_duration = set\_timer()  
  
 if timer\_duration is not None:  
 countdown\_timer(timer\_duration)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()**

**EXPLAINATION:**

**Certainly, here's a brief explanation of the provided code:**

**1. \*\*Imports:\*\***

**- `speech\_recognition`: This module provides functions for capturing audio input (voice) and recognizing it as text.**

**- `pyttsx3`: This module is used for converting text into speech.**

**- `time`: This standard Python module provides various time-related functions.**

**- `paho.mqtt.client`: This module facilitates MQTT (Message Queuing Telemetry Transport) communication, commonly used for IoT devices.**

**2. \*\*MQTT Configuration:\*\***

**- The code sets up MQTT parameters: `MQTT\_BROKER`, `MQTT\_PORT`, and `MQTT\_TOPIC`. These are used to communicate timer-related information over the MQTT protocol.**

**3. \*\*`speak` Function:\*\***

**- Converts the provided text into speech using the `pyttsx3` module.**

**4. \*\*`get\_voice\_input` Function:\*\***

**- Captures voice input from the user using a microphone.**

**- Adjusts for ambient noise, listens to audio, and uses Google's speech recognition to convert it to lowercase text.**

**- Handles exceptions for recognition errors or request errors.**

**5. \*\*`set\_timer` Function:\*\***

**- Asks the user to provide the timer duration in seconds through voice input.**

**- Converts the provided input to an integer representing the timer duration.**

**- Provides feedback to the user about the set timer.**

**6. \*\*`countdown\_timer` Function:\*\***

**- Establishes an MQTT client connection.**

**- Implements a countdown loop, starting from the provided timer duration and decrementing by one second.**

**- Prints and publishes the remaining time in the countdown to the MQTT topic.**

**- After the countdown ends, it publishes a "Timer ended" message.**

**7. \*\*`main` Function:\*\***

**- Welcomes the user and initiates the timer-setting process.**

**- Calls the `set\_timer` function to get the timer duration from the user.**

**- If a valid duration is provided, it calls the `countdown\_timer` function.**

**8. \*\*Running the Program:\*\***

**- The script starts by welcoming the user and asking for a timer duration.**

**- It captures the timer duration using voice input, validates it, and starts a countdown.**

**- During the countdown, the remaining time is displayed and published to the MQTT topic.**

**- After the countdown, a "Timer ended" message is displayed and published.**

**Overall, the code sets up a voice-controlled timer using speech recognition, displays the countdown, and communicates timer-related information using MQTT messaging.**

**stopwatch.py**

**import pyttsx3  
import speech\_recognition as sr  
import threading  
import time  
import paho.mqtt.client as mqtt  
  
MQTT\_BROKER = "broker.emqx.io"  
MQTT\_PORT = 1883  
MQTT\_TOPIC = "alarm\_system"  
  
stopwatch\_running = False  
start\_time = 0  
countdown\_thread = None  
  
  
def speak(text):  
 *"""Converts text to speech."""* engine = pyttsx3.init()  
 engine.say(text)  
 engine.runAndWait()  
def start\_stopwatch():  
 *"""Starts the stopwatch."""* global stopwatch\_running, start\_time  
 stopwatch\_running = True  
 start\_time = time.time()  
  
  
def stop\_stopwatch():  
 *"""Stops the stopwatch."""* global stopwatch\_running  
 stopwatch\_running = False  
  
  
def format\_time(seconds):  
 *"""Formats the elapsed time in HH:MM:SS format."""* m, s = divmod(int(seconds), 60)  
 h, m = divmod(m, 60)  
 return f"{h:02d}:{m:02d}:{s:02d}"  
  
def stopwatch\_timer():  
 *"""Prints the stopwatch timer in the format HH:MM:SS."""* global stopwatch\_running, start\_time  
  
 while stopwatch\_running:  
 elapsed\_time = time.time() - start\_time  
 print(f"\rStopwatch: {format\_time(elapsed\_time)}", end="")  
 send\_mqtt\_time(format\_time(elapsed\_time)) # Send the time to MQTT  
 time.sleep(1)  
  
def countdown\_thread\_function(duration):  
 *"""Starts the countdown for the given duration."""* global stopwatch\_running  
  
 while duration > 0 and stopwatch\_running:  
 time.sleep(1)  
 duration -= 1  
  
 stop\_stopwatch()  
  
def process\_voice\_command(command):  
 *"""Processes the voice command."""* global countdown\_thread, stopwatch\_running  
  
 if "start" in command:  
 if not stopwatch\_running:  
 start\_stopwatch()  
 countdown\_thread = threading.Thread(target=countdown\_thread\_function, args=(60,)) # Change the countdown duration as needed (60 seconds here).  
 countdown\_thread.start()  
 threading.Thread(target=stopwatch\_timer).start() # Start the stopwatch timer in a separate thread.  
  
 elif "stop" in command:  
 if stopwatch\_running:  
 print("stopwatch ended!!")  
 stop\_stopwatch()  
  
 elif "exit" in command:  
  
 print("Exiting the program.")  
  
 if stopwatch\_running:  
 stop\_stopwatch()  
  
 return "exit" # Return "exit" to indicate program should exit  
  
 return "continue"  
  
  
def get\_voice\_command():  
 *"""Captures voice command and returns the recognized text."""* recognizer = sr.Recognizer()  
  
 with sr.Microphone() as source:  
 recognizer.adjust\_for\_ambient\_noise(source, duration=1)  
 audio = recognizer.listen(source)  
  
 try:  
 return recognizer.recognize\_google(audio).lower()  
 except sr.UnknownValueError:  
 return ""  
 except sr.RequestError:  
 print("Sorry, I am unable to access the speech recognition service. Please try again later.")  
 return ""  
  
def send\_mqtt\_time(time\_str):  
 *"""Sends the stopwatch time to the MQTT broker."""* client = mqtt.Client()  
 client.connect(MQTT\_BROKER, MQTT\_PORT)  
 client.publish(MQTT\_TOPIC, time\_str)  
 client.disconnect()  
  
def main():  
 print("Welcome to the Voice Controlled Stopwatch.")  
 speak("Welcome to the Voice Controlled Stopwatch.")  
 speak("you can say START to start the stopwatch, STOP to stop the stopwatch and EXIT to exit the stopwatch")  
  
 while True:  
 command = get\_voice\_command()  
  
 if command:  
 result = process\_voice\_command(command)  
 if result == "exit":  
 break  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()**

**EXPLAINATION OF stopwatch.py:**

1. **Imports:**
   * **pyttsx3 is used to convert text to speech.**
   * **speech\_recognition provides voice recognition capabilities.**
   * **threading is used to manage threads for concurrent operations.**
   * **time is used for time-related functions.**
   * **paho.mqtt.client facilitates MQTT communication.**
2. **Global Variables:**
   * **stopwatch\_running: Indicates whether the stopwatch is running or not.**
   * **start\_time: Records the starting time when the stopwatch is started.**
   * **countdown\_thread: Manages the countdown thread.**
3. **speak Function:**
   * **Converts provided text into speech.**
4. **Stopwatch Functions:**
   * **start\_stopwatch: Sets stopwatch\_running to True and records the start time.**
   * **stop\_stopwatch: Stops the stopwatch by setting stopwatch\_running to False.**
   * **format\_time: Formats elapsed time into HH:MM:SS format.**
   * **stopwatch\_timer: Prints and sends the formatted elapsed time to MQTT while the stopwatch is running.**
5. **countdown\_thread\_function:**
   * **Manages the countdown for a given duration while considering both the countdown value and the stopwatch state.**
6. **Voice Command Processing:**
   * **process\_voice\_command: Processes recognized voice commands to control the stopwatch. Starts and stops the stopwatch based on the command.**
7. **Voice Recognition and MQTT Functions:**
   * **get\_voice\_command: Captures voice command and converts it to lowercase text.**
   * **send\_mqtt\_time: Publishes the stopwatch time to an MQTT topic.**
8. **main Function:**
   * **Welcomes the user and explains available voice commands.**
   * **Enters a loop where it captures and processes voice commands.**
   * **Breaks the loop and exits the program if the user says "EXIT".**
9. **Running the Program:**
   * **The script initiates by welcoming the user and explaining the available voice commands.**
   * **It listens to voice commands, processes them, and controls the stopwatch accordingly.**
   * **The stopwatch time is displayed, sent to MQTT, and the program can be exited using the "EXIT" command.**

**world\_clock.py**

**# Function to convert text to speech  
def speak(text):  
 *"""Converts text to speech."""* engine = pyttsx3.init()  
 engine.say(text)  
 engine.runAndWait()  
  
# Function to get the current time in a specific timezone  
def get\_time\_in\_timezone(timezone\_name):  
 timezone = pytz.timezone(timezone\_name)  
 current\_time = datetime.now(timezone).strftime("%H:%M")  
 return current\_time  
  
# Function to get and publish the time in a given country's timezones  
def get\_country\_time(country, client):  
 timezones = country\_timezones.get(country, [])  
 if not timezones:  
 print(f"Invalid country: {country}")  
 return  
  
 print(f"Time in {country}:")  
 for tz\_name in timezones:  
 current\_time = get\_time\_in\_timezone(tz\_name)  
 print(f"{tz\_name}: {current\_time}")  
 speak(f"{tz\_name}: {current\_time}")  
  
 # Publish the message to MQTT broker  
 client.publish(MQTT\_TOPIC, f"{country} - {current\_time}")  
# Callback function when connecting to MQTT broker  
def on\_connect(client, userdata, flags, rc):  
 print("Connected to MQTT broker with code", rc)  
  
def main():  
 speak("welcome to the world clock. here you can know the time for any country , just say the name of the country.")  
 client = mqtt.Client()  
 client.on\_connect = on\_connect  
  
 client.connect(MQTT\_BROKER\_HOST, MQTT\_BROKER\_PORT, 60)  
  
 recognizer = sr.Recognizer()  
  
 while True:  
 with sr.Microphone() as source:  
 print("Speak the country name (or say 'exit' to quit):")  
 audio = recognizer.listen(source)  
  
 try:  
 country = recognizer.recognize\_google(audio).strip().title()  
 print(f"You said: {country}")  
  
 if country == 'Exit':  
 break  
  
 get\_country\_time(country, client)  
 except sr.UnknownValueError:  
 print("Sorry, I didn't catch that. Please try again.")  
 except sr.RequestError:  
 print("Speech recognition service unavailable. Please check your internet connection.")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()**

**EXPLAINATION OF world\_clock.py:**

1. **MQTT and Timezone Definitions:**
   * **The script starts by defining MQTT broker settings, including the broker's hostname, port, and topic to publish messages.**
   * **It also contains a dictionary named country\_timezones that maps countries to their corresponding timezones.**
2. **speak Function:**
   * **Utilizes the pyttsx3 module to convert text into speech.**
3. **get\_time\_in\_timezone Function:**
   * **Takes a timezone name as input and returns the current time in that timezone.**
   * **It uses the pytz library to work with timezones and returns the time in the "HH:MM" format.**
4. **get\_country\_time Function:**
   * **Takes a country name and an MQTT client as inputs.**
   * **Retrieves the corresponding timezones from the country\_timezones dictionary.**
   * **Iterates through the timezones, gets the current time using get\_time\_in\_timezone, prints and speaks the time, and publishes it to the MQTT broker.**
5. **MQTT Callback Function on\_connect:**
   * **This function is called when the script connects to the MQTT broker.**
   * **It prints a message indicating successful connection and the connection code.**
6. **Main Function:**
   * **The main function starts by welcoming the user and explaining the functionality of the world clock.**
   * **It creates an MQTT client, sets the on\_connect callback function, and connects to the MQTT broker.**
7. **Voice Recognition and Processing Loop:**
   * **The main loop captures the user's voice input using the microphone and speech\_recognition module.**
   * **The recognized speech is processed and converted to title case (capitalizing the first letter of each word).**
   * **If the user says "exit," the loop terminates.**
   * **Otherwise, the script attempts to get and display the time for the recognized country using the get\_country\_time function.**
8. **Running the Program:**
   * **The script starts by welcoming the user to the world clock system.**
   * **It listens for voice commands using the microphone.**
   * **When a country name is recognized, it retrieves and displays the time for that country in different timezones.**
   * **The time is spoken, printed, and published to the MQTT broker.**
   * **The user can exit the program by saying "exit."**

**reminder.py**

import time  
import speech\_recognition as sr  
import pyttsx3  
import paho.mqtt.client as mqtt  
  
  
# Your MQTT broker settings  
MQTT\_BROKER\_HOST = "broker.emqx.io" # Replace with your MQTT broker's hostname or IP  
MQTT\_BROKER\_PORT = **1883** # Default MQTT port  
MQTT\_TOPIC = "alarm\_system" # MQTT topic to publish reminders  
  
# Function to convert text to speech  
def speak(text):  
 *"""Converts text to speech."""* engine = pyttsx3.init()  
 engine.say(text)  
 engine.runAndWait()  
  
# Function to recognize speech input  
def recognize\_speech(prompt):  
 recognizer = sr.Recognizer()  
  
 with sr.Microphone() as source:  
 print(prompt)  
 audio = recognizer.listen(source)  
  
 try:  
 user\_input = recognizer.recognize\_google(audio).strip()  
 print(f"You said: {user\_input}")  
 return user\_input  
 except sr.UnknownValueError:  
 print("Sorry, I didn't catch that. Please try again.")  
 return recognize\_speech(prompt)  
 except sr.RequestError:  
 print("Speech recognition service unavailable. Please check your internet connection.")  
 return None  
  
# Function to convert text to speech  
def speak(text):  
 engine = pyttsx3.init()  
 engine.say(text)  
 engine.runAndWait()  
  
# Function to get reminder text from user  
def get\_reminder\_text():  
 speak("Please say your reminder:")  
 return recognize\_speech("Please say your reminder:")  
  
# Function to get reminder time from user  
def get\_reminder\_time():  
 speak("Please say the hour (0-23) for the reminder: ")  
 hour = int(recognize\_speech("Please say the hour (0-23) for the reminder: "))  
 speak("Please say the minute (0-59) for the reminder: ")  
 minute = int(recognize\_speech("Please say the minute (0-59) for the reminder: "))  
 return hour**,** minute  
  
# Callback function when connecting to MQTT broker  
def on\_connect(client**,** userdata**,** flags**,** rc):  
 print("Connected to MQTT broker with code"**,** rc)  
  
def main():  
 speak("welcome of the reminder system.")  
 client = mqtt.Client()  
 client.on\_connect = on\_connect  
  
 client.connect(MQTT\_BROKER\_HOST**,** MQTT\_BROKER\_PORT**, 60**)  
  
 while True:  
 reminder\_text = get\_reminder\_text()  
 if reminder\_text is not None:  
 hour**,** minute = get\_reminder\_time()  
 current\_time = time.localtime()  
 reminder\_time = time.struct\_time((current\_time.tm\_year**,** current\_time.tm\_mon**,** current\_time.tm\_mday**,** hour**,** minute**, 0,** current\_time.tm\_wday**,** current\_time.tm\_yday**,** current\_time.tm\_isdst))  
 reminder\_timestamp = time.mktime(reminder\_time)  
  
 now = time.mktime(current\_time)  
 wait\_time = reminder\_timestamp - now  
 if wait\_time <= **0**:  
 speak("Reminder time is in the past. Please set a future time.")  
 else:  
 print(f"Reminder set for: {hour:02d}:{minute:02d}")  
 speak(f"Reminder set for: {hour:02d}:{minute:02d}")  
 time.sleep(wait\_time)  
 speak(f"Reminder: {reminder\_text}")  
 final\_reminder = f"Reminder: {reminder\_text} - Time: {hour:02d}:{minute:02d}"  
 client.publish(MQTT\_TOPIC**,** final\_reminder)  
  
 speak("Do you want to set another reminder? (yes/no): ")  
 exit\_choice = recognize\_speech("Do you want to set another reminder? (yes/no): ").lower()  
 if exit\_choice != "yes":  
 break  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**EXPLAINATION OF reminder.py:**

1. **MQTT and User Input Settings:**
   * **The script starts by defining MQTT broker settings, including the broker's hostname, port, and topic for publishing reminders.**
2. **speak Function:**
   * **Utilizes the pyttsx3 module to convert text into speech.**
3. **recognize\_speech Function:**
   * **Utilizes the speech\_recognition module to capture and recognize voice input.**
   * **Captures audio using the microphone and uses Google's speech recognition service to convert it to text.**
   * **Handles recognition errors and request errors, prompting the user to try again if needed.**
4. **get\_reminder\_text Function:**
   * **Instructs the user to provide a reminder and uses the recognize\_speech function to capture the reminder text.**
5. **get\_reminder\_time Function:**
   * **Instructs the user to provide the hour and minute for the reminder time.**
   * **Uses recognize\_speech to capture and convert the input to integers.**
6. **MQTT Callback Function on\_connect:**
   * **This function is called when the script connects to the MQTT broker.**
   * **It prints a message indicating a successful connection and the connection code.**
7. **Main Function:**
   * **The main function starts by welcoming the user to the reminder system using text-to-speech.**
   * **It creates an MQTT client, sets the on\_connect callback function, and connects to the MQTT broker.**
8. **Reminder Setup and Publishing Loop:**
   * **The main loop captures reminder text and time using get\_reminder\_text and get\_reminder\_time functions.**
   * **It calculates the time remaining until the reminder and checks if it's set in the past.**
   * **If not in the past, it sets the reminder, waits until the reminder time, speaks the reminder, and publishes it to the MQTT broker.**
9. **Continuation Check:**
   * **After handling a reminder, the script prompts the user to set another reminder or exit.**
   * **The user's choice is captured using recognize\_speech, and if the choice is not "yes," the loop terminates.**
10. **Running the Program:**
    * **The script starts by welcoming the user to the reminder system.**
    * **It listens for voice commands, capturing reminder text and time.**
    * **If the reminder time is set for the future, the script waits until the reminder time and then speaks the reminder text.**
    * **The reminder information is published to the MQTT broker.**
    * **The user can set multiple reminders or exit the program by saying "yes" or "no," respectively.**

**alarm.py**

**import time  
import pyttsx3  
import paho.mqtt.client as mqtt  
import speech\_recognition as sr  
  
# Your MQTT broker settings  
MQTT\_BROKER\_HOST = "broker.emqx.io" # Replace with your MQTT broker's hostname or IP  
MQTT\_BROKER\_PORT = 1883 # Default MQTT port  
MQTT\_TOPIC = "alarm\_system" # MQTT topic to publish alarm commands  
  
  
def speak(text):  
 *"""Converts text to speech."""* engine = pyttsx3.init()  
 engine.say(text)  
 engine.runAndWait()  
  
  
def recognize\_time\_input(prompt):  
 recognizer = sr.Recognizer()  
  
 while True:  
 try:  
 speak(prompt)  
 with sr.Microphone() as source:  
 audio = recognizer.listen(source)  
 time\_input = recognizer.recognize\_google(audio).lower()  
  
 value = parse\_hour\_minute(time\_input)  
 if value is not None:  
 return value  
 else:  
 speak("Invalid time input. Please try again.")  
 except sr.UnknownValueError:  
 speak("Sorry, I didn't catch that. Please try again.")  
 except sr.RequestError:  
 speak("Sorry, I am unable to access the speech recognition service. Please try again later.")  
 except ValueError:  
 speak("Invalid input. Please provide a valid number.")  
  
  
def parse\_hour\_minute(time\_input):  
 try:  
 value = int(time\_input)  
 if 0 <= value <= 59:  
 return value  
 except ValueError:  
 pass  
 return None  
  
  
def on\_connect(client, userdata, flags, rc):  
 print("Connected to MQTT broker with code", rc)  
  
  
def main():  
 client = mqtt.Client()  
 client.on\_connect = on\_connect  
  
 client.connect(MQTT\_BROKER\_HOST, MQTT\_BROKER\_PORT, 60)  
  
 speak("Welcome to the Alarm System. Set time for the alarm.")  
  
 while True:  
 alarm\_hour = recognize\_time\_input("Please say the alarm hour.")  
 alarm\_minute = recognize\_time\_input("Please say the alarm minute.")  
  
 current\_time = time.localtime()  
 alarm\_time = time.struct\_time((current\_time.tm\_year, current\_time.tm\_mon, current\_time.tm\_mday, alarm\_hour,  
 alarm\_minute, 0, current\_time.tm\_wday, current\_time.tm\_yday,  
 current\_time.tm\_isdst))  
 alarm\_timestamp = time.mktime(alarm\_time)  
  
 now = time.mktime(current\_time)  
 wait\_time = alarm\_timestamp - now  
  
 if wait\_time <= 0:  
 speak("Alarm time is in the past. Please set a future time.")  
 else:  
 alarm\_time\_str = f"Alarm time: {alarm\_hour:02d}:{alarm\_minute:02d}"  
 print(alarm\_time\_str)  
 client.publish(MQTT\_TOPIC, alarm\_time\_str)  
  
 time.sleep(wait\_time)  
  
 alarm\_alert\_message = "ALARM ALERT !!"  
 client.publish(MQTT\_TOPIC, alarm\_alert\_message)  
  
 speak("Wake up! The alarm is ringing.")  
 break # Exit the loop after setting the alarm  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()**

**EXPLAINATION OF alarm.py:**

1. **MQTT and Voice Recognition Settings:**
   * **The script begins by defining MQTT broker settings, including the broker's hostname, port, and topic for publishing alarm commands.**
2. **speak Function:**
   * **Utilizes the pyttsx3 module to convert text into speech.**
3. **recognize\_time\_input Function:**
   * **Utilizes the speech\_recognition module to capture and recognize voice input.**
   * **Captures audio using the microphone and uses Google's speech recognition service to convert it to text.**
   * **Validates and returns time input (hours and minutes) recognized from speech.**
   * **If the recognized input is invalid, the user is prompted to try again.**
4. **parse\_hour\_minute Function:**
   * **Converts recognized time input to an integer and checks if it's within the valid range of 0 to 59.**
   * **If the input is a valid minute value, it's returned; otherwise, None is returned.**
5. **MQTT Callback Function on\_connect:**
   * **This function is called when the script connects to the MQTT broker.**
   * **It prints a message indicating successful connection and the connection code.**
6. **Main Function:**
   * **The main function creates an MQTT client and sets the on\_connect callback function.**
   * **It connects to the MQTT broker and welcomes the user to the Alarm System using text-to-speech.**
7. **Alarm Setup and Alert Loop:**
   * **The main loop captures the alarm hour and minute using recognize\_time\_input function.**
   * **It calculates the alarm time and wait time until the alarm using time-related functions.**
   * **If the alarm time is set in the past, the script informs the user to set a future time.**
   * **Otherwise, the alarm time is published to the MQTT broker as a message and a corresponding time.**
   * **The script then waits until the alarm time and publishes an alarm alert message.**
   * **Finally, the user is alerted with a wake-up message, and the loop exits.**
8. **Running the Program:**
   * **The script starts by connecting to the MQTT broker and welcoming the user.**
   * **It captures the alarm hour and minute through voice recognition.**
   * **If the provided time is valid and in the future, the script sets the alarm and waits until the alarm time.**
   * **When the alarm time arrives, an alarm alert message is published, and the user is alerted with a wake-up message.**

**current\_time.py**

**import pyttsx3  
import speech\_recognition as sr  
import time  
import paho.mqtt.client as mqtt  
  
# Your MQTT broker settings  
MQTT\_BROKER\_HOST = "broker.emqx.io" # Replace with your MQTT broker's hostname or IP  
MQTT\_BROKER\_PORT = 1883 # Default MQTT port  
MQTT\_TOPIC = "alarm\_system" # MQTT topic to publish current time  
  
def speak(text):  
 *"""Converts text to speech."""* engine = pyttsx3.init()  
 engine.say(text)  
 engine.runAndWait()  
  
def get\_voice\_input():  
 *"""Captures voice input and returns the recognized text."""* recognizer = sr.Recognizer()  
  
 with sr.Microphone() as source:  
 recognizer.adjust\_for\_ambient\_noise(source, duration=1)  
 audio = recognizer.listen(source)  
  
 try:  
 return recognizer.recognize\_google(audio).lower()  
 except sr.UnknownValueError:  
 return ""  
 except sr.RequestError:  
 speak("Sorry, I am unable to access the speech recognition service. Please try again later.")  
 return ""  
  
def tell\_time():  
 current\_time = time.localtime()  
 hour = current\_time.tm\_hour  
 minute = current\_time.tm\_min  
 period = "AM" if hour < 12 else "PM"  
 hour %= 12  
 if hour == 0:  
 hour = 12  
 time\_str = f"current time-{hour}:{minute:02d} {period}."  
 return time\_str  
  
def on\_connect(client, userdata, flags, rc):  
 print("Connected to MQTT broker with code", rc)  
  
def main():  
 client = mqtt.Client()  
 client.on\_connect = on\_connect  
  
 client.connect(MQTT\_BROKER\_HOST, MQTT\_BROKER\_PORT, 60)  
  
 speak("Welcome! You can ask me 'What's the time?' to know the current time.")  
  
 user\_input = get\_voice\_input()  
  
 if "what's the time" in user\_input:  
 time\_str = tell\_time()  
 print(time\_str)  
 speak(time\_str)  
 client.publish(MQTT\_TOPIC, time\_str)  
  
 #speak("Exiting the program. Goodbye!")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()**

**EXPLAINATION OF current­\_time.py:**

1. **MQTT and Voice Recognition Settings:**
   * **The script starts by defining MQTT broker settings, including the broker's hostname, port, and topic for publishing current time information.**
2. **speak Function:**
   * **Utilizes the pyttsx3 module to convert text into speech.**
3. **get\_voice\_input Function:**
   * **Utilizes the speech\_recognition module to capture and recognize voice input.**
   * **Captures audio using the microphone and uses Google's speech recognition service to convert it to text.**
   * **Handles recognition errors and request errors, returning the recognized text in lowercase.**
4. **tell\_time Function:**
   * **Retrieves the current time using time.localtime().**
   * **Formats the time into the 12-hour format with AM/PM indication.**
   * **Returns a string indicating the current time in the format "current time-HH:MM AM/PM."**
5. **MQTT Callback Function on\_connect:**
   * **This function is called when the script connects to the MQTT broker.**
   * **It prints a message indicating successful connection and the connection code.**
6. **Main Function:**
   * **The main function creates an MQTT client and sets the on\_connect callback function.**
   * **It connects to the MQTT broker and welcomes the user using text-to-speech.**
7. **Time Inquiry and Publishing:**
   * **The script captures voice input from the user using get\_voice\_input.**
   * **If the recognized input contains the phrase "what's the time," the script retrieves the current time using tell\_time.**
   * **The current time is printed, spoken, and published to the MQTT broker.**
8. **Running the Program:**
   * **The script connects to the MQTT broker and welcomes the user.**
   * **It listens for voice input to inquire about the current time.**
   * **If the user asks for the time, the script responds with the current time and publishes it to the MQTT broker.**

**Using Arduino connecting MQTT with Wiznet :**

**Source code with Explanation:**

**#include "HardwareSerial.h"**

**#include <SPI.h>**

**#include <Wire.h>**

**#include <SoftWire.h>**

**#include <Adafruit\_GFX.h>**

**#include <Adafruit\_SSD1306.h>**

**#include <Ethernet.h>**

**#include <PubSubClient.h>**

**// #define PIN0\_SDA 18**

**// #define PIN0\_SCL 19**

**#define SCREEN\_WIDTH 128**

**#define SCREEN\_HEIGHT 32**

**#define OLED\_RESET -1**

**#define SCREEN\_ADDRESS 0x3C**

**byte mac[] = { 0xDE, 0xAD, 103, 0xEF, 0xFE, 0xED }; // Replace with your desired MAC address**

**IPAddress ip(172, 16, 0, 100); // Replace with the desired IP address**

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

**const char\* alarm\_system = "alarm\_system" ;**

**EthernetClient ethClient ;**

**PubSubClient client( ethClient );**

**Adafruit\_SSD1306 display(SCREEN\_WIDTH, SCREEN\_HEIGHT, &Wire, OLED\_RESET);**

**void setup() {**

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

**Serial3.setRx(PC11);**

**Serial3.setTx(PC10);**

**Serial3.begin(9600);**

**while (!Serial) {**

**; // wait for serial port to connect. Needed for native USB port only**

**}**

**Wire.setSDA(14);**

**Wire.setSCL(15);**

**Wire.begin();**

**if (!display.begin(SSD1306\_SWITCHCAPVCC, SCREEN\_ADDRESS)) {**

**for (;;) {**

**}**

**}**

**display.clearDisplay();**

**display.display();**

**// Print "CLOCK" on the OLED display**

**display.setTextSize(2);      // Set text size**

**display.setTextColor(SSD1306\_WHITE); // Set text color**

**display.setCursor(0, 0);     // Set text cursor position**

**display.println("SMART ");    // Print "CLOCK"**

**display.setTextSize(2);      // Set text size**

**display.println("     CLOCK ");**

**display.display();           // Display the content**

**// Ethernet setup**

**Ethernet.begin(mac);**

**//Ethernet.init(17);**

**// Wait for Ethernet to be ready**

**while (!Ethernet.begin(mac)) {**

**Serial.println("Failed to configure Ethernet using DHCP");**

**delay(10000);**

**}**

**Serial.print("Ethernet connected! IP address: ");**

**Serial.println(Ethernet.localIP());**

**delay(2000);**

**// MQTT setup**

**client.setServer(server , 1883);**

**client.setCallback(callback);**

**reconnect();**

**client.subscribe("alarm\_system"); // Replace with your MQTT topic for timer duration**

**}**

**void loop() {**

**if (!client.connected()) {**

**reconnect();**

**}**

**client.loop();**

**}**

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

**String message = "";**

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

**message += (char)payload[i];**

**}**

**Serial.print("Received message on topic: ");**

**Serial.println(topic);**

**Serial.print("Message: ");**

**Serial.println(message);**

**// Clear the OLED display**

**display.clearDisplay();**

**// Display received message on the OLED screen**

**display.setTextSize(1.5);      // Set text size**

**display.setTextColor(SSD1306\_WHITE); // Set text color**

**display.setCursor(0, 0);     // Set text cursor position**

**//display.println("Received:"); // Print a label**

**display.println(message);     // Print the received message**

**display.display();           // Display the content**

**}**

**void reconnect() {**

**while (!client.connected()) {**

**Serial.println("Attempting MQTT connection...");**

**if (client.connect("arduinoClient103")) {**

**Serial.println("connected");**

**client.subscribe(alarm\_system); // Subscribe to the MQTT topic for timer duration**

**} else {**

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

**Serial.print(client.state());**

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

**delay(5000);**

**}**

**}**

**}**

**Explanation:**

This code is an Arduino sketch that sets up a system to display messages received from an MQTT broker on an SSD1306 OLED display and establish an Ethernet connection for MQTT communication. Let's break down the different parts of the code:

1. \*\*Libraries and Definitions:\*\*

- The code includes several libraries, such as `HardwareSerial`, `SPI`, `Wire`, `SoftWire`, `Adafruit\_GFX`, `Adafruit\_SSD1306`, `Ethernet`, and `PubSubClient`.

- Some constants are defined: `SCREEN\_WIDTH` and `SCREEN\_HEIGHT` for the dimensions of the OLED display, `OLED\_RESET` for the reset pin of the OLED (set to -1), and `SCREEN\_ADDRESS` for the I2C address of the OLED.

- MQTT-related constants are defined, such as `MQTT\_BROKER` (the MQTT broker's address), `MQTT\_PORT` (the MQTT broker's port), and `CLIENT\_ID` (the client's unique identifier).

- MAC address and IP address for Ethernet connection are defined.

2. \*\*Global Variables:\*\*

- `ethClient` is an instance of the Ethernet client for communication over the Ethernet connection.

- `client` is an instance of the PubSubClient library, used for MQTT communication.

3. \*\*Setup Function (`void setup()`):\*\*

- The code sets up serial communication and waits for a serial connection to be established.

- Wire library's SDA and SCL pins are configured.

- The OLED display is initialized using the `display.begin()` function. If the display initialization fails, the program enters an infinite loop.

- The OLED display is cleared, and the text "SMART CLOCK" is displayed in large text.

- Ethernet connection is established using `Ethernet.begin(mac)`. It attempts to configure the Ethernet connection using DHCP and waits until it succeeds.

- After successful Ethernet connection, the Arduino's local IP address is printed.

- MQTT client is configured to connect to the MQTT broker, and the callback function `callback` is set for handling received MQTT messages. The `reconnect()` function is called to establish the MQTT connection and subscribe to the "alarm\_system" topic.

4. \*\*Loop Function (`void loop()`):\*\*

- The loop checks if the MQTT client is connected. If not, it calls the `reconnect()` function to attempt reconnection.

- The MQTT client's loop function is called to maintain MQTT communication.

5. \*\*Callback Function (`void callback(char\* topic, byte\* payload, unsigned int length)`:\*\*

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

- The received message's payload is converted into a String.

- The received topic and message are printed to the serial monitor.

- The OLED display is cleared, and the received message is displayed on the screen.

6. \*\*Reconnect Function (`void reconnect()`):\*\*

- This function attempts to connect to the MQTT broker.

- It prints a status message and tries to connect the client with a specific client ID.

- If the connection is successful, the client subscribes to the "alarm\_system" topic.

- If the connection fails, the client's state is printed, and a delay of 5 seconds is introduced before retrying.

The main purpose of this code is to display messages received from an MQTT broker on an OLED display and establish a stable connection to the MQTT broker for communication. It integrates Ethernet communication, MQTT messaging, and OLED display functionality into a single Arduino sketch.