Voice-Controlled Smart Lighting System for Smart Home

Bang Nguyen

Content

1. Motivation for Topic Selection

In the era of the Fourth Industrial Revolution, the race for emerging technologies such as Artificial Intelligence (AI) and the Internet of Things (IoT) has been accelerating at an unprecedented pace. The emergence of these technologies has been, is, and will continue to profoundly transform human life.

The Internet of Things (IoT) is a network that connects devices with each other via the Internet. With the emergence of IoT, the scenario of a world in which every object is assigned a unique identifier and all objects are capable of transmitting and communicating with one another through a unified network without the need for human intervention is gradually becoming a reality. IoT technology has been bringing greater convenience and new experiences to people in their daily lives. Today, IoT is increasingly becoming one of the most prominent and significant technologies of the 21st century, with the potential to revolutionize various fields such as healthcare, agriculture, and industry.

Through research and group discussions, we have identified the topic "Design of a Voice Controlled Smart Lighting System for Smart Homes and Smart Farms" as both interesting and well-suited to our team's capabilities. This report focuses on the design and development of a smart lighting system based on voice recognition technology and Internet-based communication. By utilizing voice recognition, the system allows users to control lighting such as turning lights on/off or scheduling their operation using only a few simple voice commands. This voice activated system is expected to deliver a modern and immersive user experience while also contributing to energy savings and environmental protection.

2. General Overview

2.1. Description of System Functions

The project titled "Voice-Controlled Smart Lighting System for Smart Home" focuses on developing a system that enables users to control LED lights using voice commands without the need for traditional switches or remote controllers. Instead of

searching for and pressing on/off buttons on a control panel or using a remote control, users can simply issue voice commands to operate the LED lighting system.

The lighting system operates in two modes: automatic mode and timer mode.

- Automatic mode: The system can turn the lights on or off when it detects the presence of people in the area.

- Timer mode: The system can control turning the lights on or off within a fixed time period set by the user.

To achieve this, the project requires the integration of voice recognition technology and natural language processing, enabling the ESP32 microcontroller to process and transmit data via UART communication to the Arduino Nano, which then controls the LED lights.

2.2. Introduction to the MQTT Protocol

❖ Definition:

MQTT (short for *Message Queuing Telemetry Transport*) is a messaging protocol based on the publish/subscribe model. It is designed for Internet of Things (IoT) devices that operate with low bandwidth, require high reliability, and need to function over unstable or constrained networks. MQTT relies on a lightweight broker that acts as an intermediary server, and it is designed to be open (i.e., not tailored to any specific application), simple, and easy to implement.

❖ Ideal Environment:

MQTT is an ideal choice in the following environments:

- In scenarios where network connectivity is expensive, bandwidth is limited, or the connection is frequently unstable, MQTT provides a robust and efficient solution for maintaining reliable data communication.

- For embedded systems that operate under strict hardware constraints such as limited processing power, reduced memory capacity, and low energy availability MQTT's lightweight design allows for smooth integration and dependable performance.

- Due to its low bandwidth consumption and ability to operate effectively in high-latency environments, MQTT is exceptionally well-suited for Machine-to-Machine (M2M) communication, where reliability and responsiveness are critical.

❖ History and Development:

MQTT was invented by Andy Stanford-Clark (IBM) and Arlen Nipper (Eurotech) in late 1999, when their task was to create a communication protocol that minimized power consumption and bandwidth usage for connecting oil pipelines via satellite links.

- In 2011, IBM and Eurotech contributed MQTT to an Eclipse Foundation project called *Paho*.

- In 2013, MQTT was submitted to OASIS (Organization for the Advancement of Structured Information Standards) for standardization.

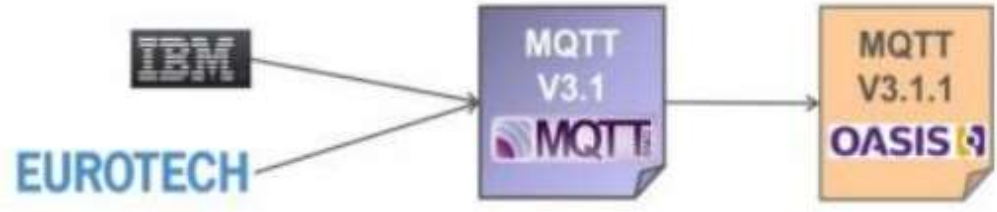

Figure 2. 1 Evolution of the MQTT Protocol: From IBM to OASIS Standardization.

2.3 Overview of the IoT Architecture

❖ The Role of MQTT in the IoT Architecture:

Some notable advantages of MQTT include low bandwidth, high reliability, and the ability to operate even in unstable network conditions. It uses very few bytes for server connection, and the connection can remain open throughout. It also supports connecting multiple devices (MQTT clients) through a single MQTT server (broker). Because this protocol uses low bandwidth in high-latency environments, it is an ideal protocol for IoT applications.

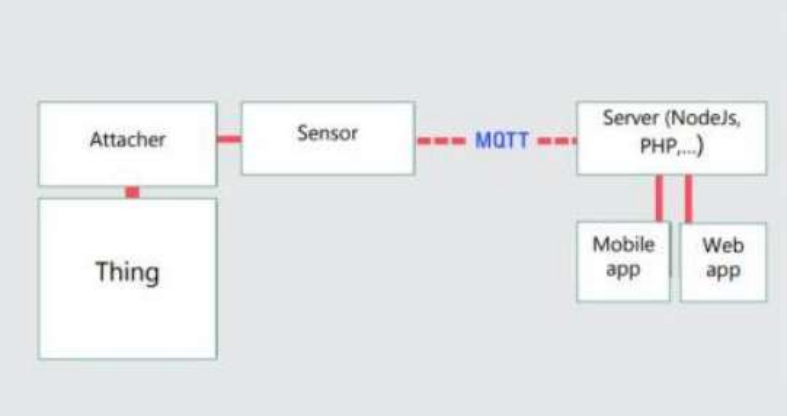

Figure 2. 2 MQTT Communication Architecture in IoT.

❖ Features and Key Characteristics:

The publish/subscribe messaging model provides one-way distributed communication that is decoupled from the application layer.

Messages are transmitted promptly and efficiently through the network, with the protocol focusing solely on the delivery mechanism rather than the specific content or meaning of the data being transmitted.

TCP/IP is used as the underlying transport protocol.

There are three levels of Quality of Service (QoS) for data transmission:

- QoS 0: The broker/client sends the message exactly once, with delivery confirmed only by the TCP/IP protocol.

- QoS 1: The broker/client ensures that the message is delivered at least once, meaning multiple duplicates may occur.

- QoS 2: The broker/client guarantees that the message is received exactly once by the recipient, involving a four-step handshake process.

The data payload encapsulation is minimal and reduced to the bare essentials to decrease transmission overhead.

❖ Advantages of MQTT:

With the features and key characteristics mentioned above, MQTT offers numerous benefits, especially in SCADA (Supervisory Control and Data Acquisition) systems for accessing IoT data:

- Enables more efficient information transmission.

- Enhances scalability.

- Significantly reduces network bandwidth consumption.

- Highly suitable for control and monitoring applications.

- Maximizes the utilization of available bandwidth.

- Low cost.

- Provides strong security and data protection.

- Widely adopted in industries such as oil and gas, and by major companies like Amazon and Facebook.

- Saves development time.

- The publish/subscribe protocol collects more data while consuming less bandwidth compared to traditional protocols.

❖ Disadvantages of MQTT:

Although MQTT possesses many notable advantages, it also has certain limitations that need to be considered when deploying it in IoT applications:

- Lack of end-to-end encryption.

- Requires additional security solutions to ensure data protection.

- Does not provide a built-in mechanism to fully guarantee that the data delivery is acknowledged by the recipient, which may lead to uncertainty about whether the message has been successfully received.

- Potential latency issues in data transmission, especially in large-scale IoT networks with numerous connected devices.

- Lower performance when applied to complex IoT systems.

- Managing connections for a large number of devices in an IoT network can become complicated when using MQTT.

- Standardization of MQTT library usage and compatibility across all devices is necessary to ensure interoperability.

2.4. Publish/Subscribe Model and MQTT Operating Mechanism

❖ Publish/Subscribe Model:

Components:

- Client.

- Publisher: The entity that sends messages.

- Subscriber: The entity that receives messages.

- Broker: The intermediary server.

Among these, the Broker is considered the central component, acting as the hub that connects all clients (both Publishers and Subscribers). The primary function of the Broker is to receive messages from Publishers, queue them, and then forward them to the appropriate destinations. Additionally, the Broker may provide supplementary features related to communication, such as message security, storage, logging, and more.

Clients are divided into two groups: Publishers and Subscribers. A client performs at least one of these two roles — either publishing messages to one or more specific topics or subscribing to one or more topics to receive messages from those topics.

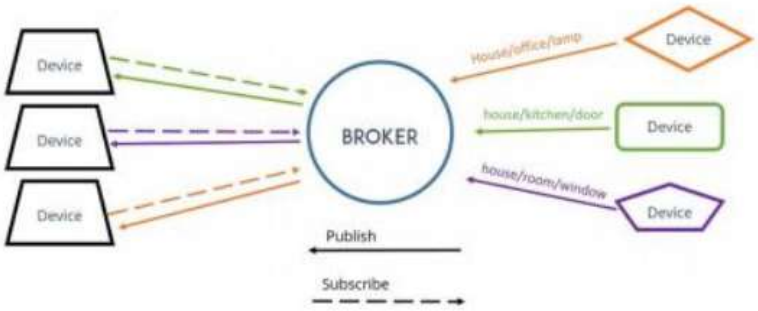

Figure 2. 3 Pub/Sub Model for IoT Devices with Broker.

MQTT clients are compatible with most existing operating system platforms, including macOS, Windows, Linux, Android, iOS, and more.

❖ **Advantages:**

- Separate, independent connections.

- Scalability.

- Time decoupling.

- Synchronization decoupling.

❖ **Disadvantages:**

- The intermediary server (Broker) does not provide acknowledgments regarding the status of message delivery. Therefore, there is no way to determine whether a message has been successfully sent or not.

- Publishers have no knowledge of the status of Subscribers, and vice versa. This raises concerns about how to ensure that everything operates correctly.

- Malicious publishers can send harmful messages, and subscribers might receive content they should not have access to.

2.5. MQTT Operating Mechanism Based on the Publish/Subscribe Model

❖ **Characteristics and Distinct Features:**

Characteristics:

- Space decoupling.

- Time decoupling.

- Synchronization decoupling.

Distinct Features:

- MQTT uses a subject-based message filtering mechanism.

- MQTT includes a layer called Quality of Service (QoS), which helps to easily determine whether a message has been successfully delivered or not.

❖ Overview of the Mechanism:

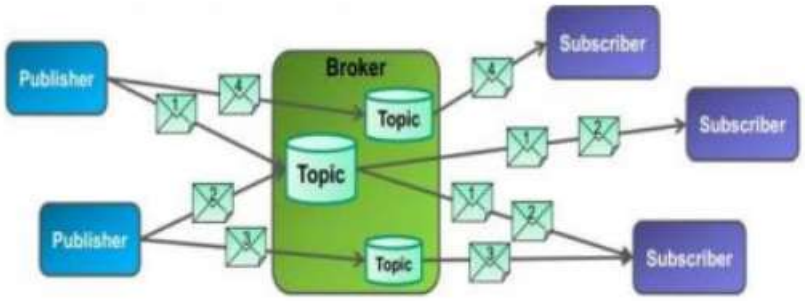

Figure 2. 4 Publisher-Subscriber Model in Messaging Systems.

MQTT operates based on a client/server mechanism, where each sensor acts as a client connecting to a server, which can be understood as an intermediary server called a Broker, via the TCP (Transmission Control Protocol). The Broker is responsible for coordinating all messages from the publishers to the appropriate subscribers.

MQTT is a message-oriented protocol. Each message is a discrete packet of data that the Broker cannot interpret. Each message is published to an address, which can be understood as a channel (Topic). Clients subscribe to one or more channels to receive or send data. A client can subscribe to multiple channels. Each client receives data whenever any other node publishes data to the subscribed channel. When a client sends a message to a certain channel, this action is called publishing.

2.6. Architecture of System Components

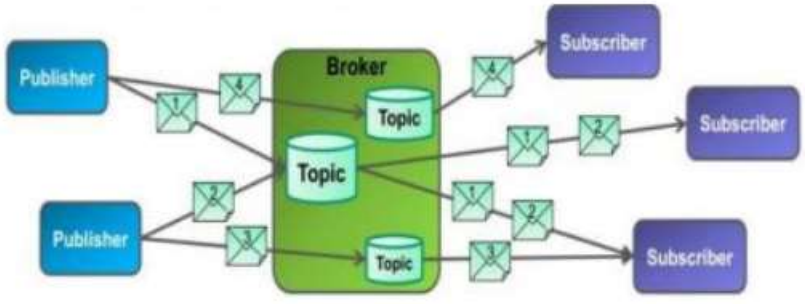

Figure 2. 5 Publisher-Subscriber Architecture with Topics and Broker.

Main components of MQTT:

- MQTT Client (Publisher/Subscriber): Clients subscribe to one or more topics to send and receive messages corresponding to those topics.

- MQTT Server (Broker): The Broker receives subscription information from clients, accepts messages published by clients, and forwards those messages to the respective subscribers based on their subscriptions.

- Topic: A Topic can be considered as a queue of messages, serving as a predefined template for Subscribers or Publishers. Logically, topics provide clients with a way to exchange information with predefined semantics. For example, temperature sensor data from a building.

- Session: A session is defined as the connection between a client and the server. All communication between the client and server occurs within the scope of a session.

- Subscription: Unlike a session, a subscription logically represents a client's connection to a topic. Once a client subscribes to a topic, it can send and receive messages associated with that topic.

2.7. Message Format

MQTT messages contain a fixed-length field (2 bytes), while the overall message length varies depending on the message type and its value.

Variable-length messages often complicate the processing within the protocol.

However, MQTT is optimized for networks with limited bandwidth or unreliable connections (often wireless networks), so variable-length messages are used to minimize data transmission as much as possible.

MQTT uses network byte order (big-endian) for bit sequencing.

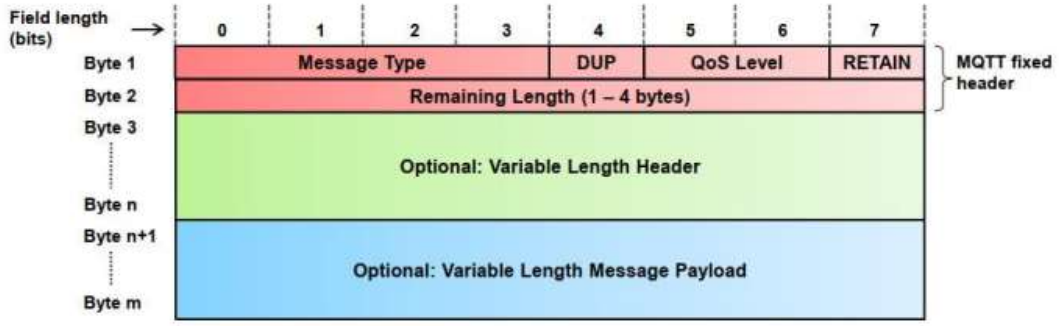

Figure 2. 6 MQTT Message Format Structure.

2.8. MQTT Quality of Service

Figure 2. 7 MQTT Quality of Service (QoS) Levels and Message Delivery Guarantees.

- At QoS level 0 (At-most-once), MQTT delivers messages on a best-effort basis without acknowledgment, relying solely on the reliability of the underlying TCP/IP network.

- At QoS level 1 (At-least-once), messages are guaranteed to be delivered, but they might be delivered more than once in case of retransmission.

- At QoS level 2 (Exactly-once), messages are guaranteed to be delivered exactly once, combining the reliability of QoS 1 and avoiding message duplication.

2.9. Practical Applications

- Can be effectively used in wet environments such as bathrooms, restrooms, and other moisture-prone areas, significantly reducing the necessity for users to physically touch switches with wet hands, thereby enhancing overall electrical safety and minimizing the risk of electric shock.

- Well suited for lighting applications in agricultural settings including farms, livestock facilities, and crop cultivation areas, offering an energy-efficient solution that helps reduce power consumption while still ensuring optimal lighting conditions to maintain and potentially improve productivity.

- Ideal for providing illumination in hard-to-access or remote locations where installing conventional switches is challenging, costly, or impractical, offering a convenient, maintenance-free lighting solution that eliminates the need for additional wiring or manual intervention.

3. Technical Factors

3.1. Technical Requirements

- The system must have a response time of less than 10 seconds.

- Detect human presence within a radius of 6 meters (automatic mode).

- The system must recognize the user's voice from a minimum distance of 10 meters.

- The timing deviation in timer mode must be less than 3 seconds.

3.2. System Block Diagram

- The user issues commands to control the lights via a laptop connected to WiFi.

- The laptop’s microphone captures the commands as audio, which are then converted to text using a Speech-to-Text algorithm.

- Once converted into text, the commands are sent to the MQTT server. From the MQTT server, the commands are transmitted to the ESP32 via WiFi.

- The ESP32 receives the text commands sent from MQTT and forwards them to the Arduino Nano via UART communication for processing.

- Upon receiving commands from the ESP32, the Arduino Nano analyzes the instructions to control the light on/off functions according to user requests and activates the corresponding status indicator lights. The command analysis algorithm of the Arduino Nano will be detailed in the following section.

- The operation of the smart voice-controlled timer light system is illustrated in the block diagram below.

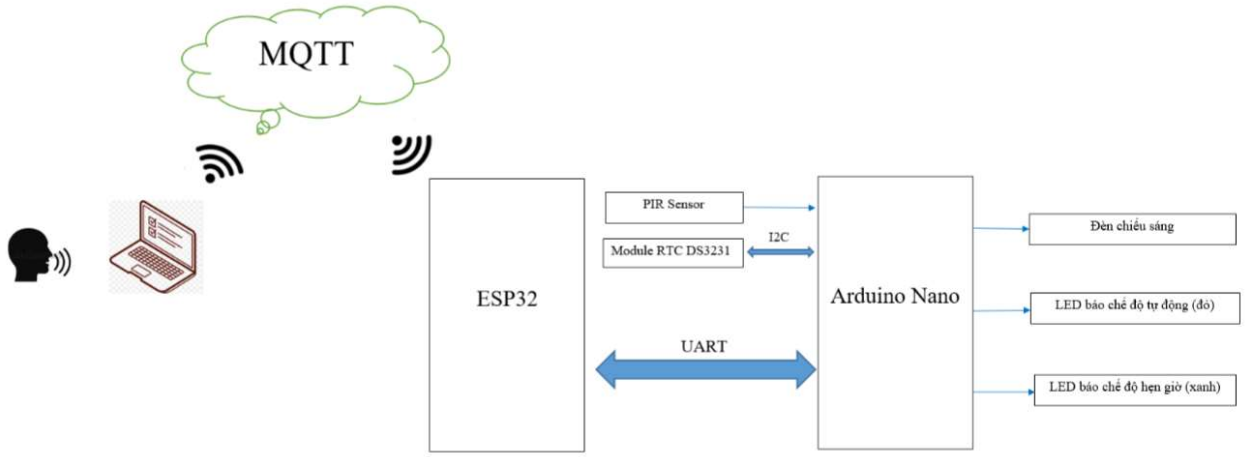

Figure 3. 1 Block Diagram of Voice-Controlled Smart Lighting System for Smart Home.

3.3. Speech Recognition Process

To perform voice recognition and convert speech to text before sending it to MQTT, we use Python along with the SpeechRecognition, paho.mqtt, and PyAudio libraries.

- First, the program establishes a connection with the MQTT broker to send data.

- Next, the program waits to capture voice input through the microphone (Vietnamese speech), then converts it into Vietnamese text without diacritics. If no command is detected, the program...

3.4. Algorithm Flowchart

When the system is powered on, it will enter the default mode, which is the automatic mode (indicated by the red LED). Then, the Arduino initializes two communication protocols: UART and I2C. UART is used for communication between the Arduino Nano and the ESP32, while I2C is used for communication between the RTC module and the Arduino.

After initialization, the Arduino enters a loop waiting to receive commands from the ESP32. Upon receiving a command from the ESP32, the Arduino checks whether it is a mode setting command or a control command.

- If it is a mode setting command, the Arduino sets the light mode, turns on the corresponding mode indicator LED, and returns to the waiting loop.

- If it is a control command, the Arduino controls the light on/off according to the command. However, if the light is currently in automatic mode, timer control commands will not take effect. Therefore, the system must check whether it is in automatic mode before executing the control commands to ensure proper operation.

Lighting Control Algorithm for Specific Modes:

Automatic mode:

- Arduino will read the value from the sensor, then determine whether a person is present or not. If a person is detected, the light will be turned on. If no person is detected, the system will wait for about 10 seconds to verify if someone is actually present. If no presence is detected after that, the light will be turned off.

Timer Mode:

- Arduino will analyze the command request. If the command is to turn off the light, it will proceed to turn off the light and return to the loop to wait for the next command.

- If the command is to turn on the light, Arduino will turn on the light, then analyze the time specified in the command and calculate and set the timer for the RTC module. Once the RTC module is configured and started, Arduino will continuously read the current time from the RTC every 0.2 seconds and compare it with the user-defined target time. If the required time has not yet been reached, the system will continue reading in the next cycle. When the specified time is reached, the system will turn off the light and return to the loop to wait for the next command.

In summary, the algorithm of the voice-controlled smart lighting timer system can be simply illustrated by the following flowchart.

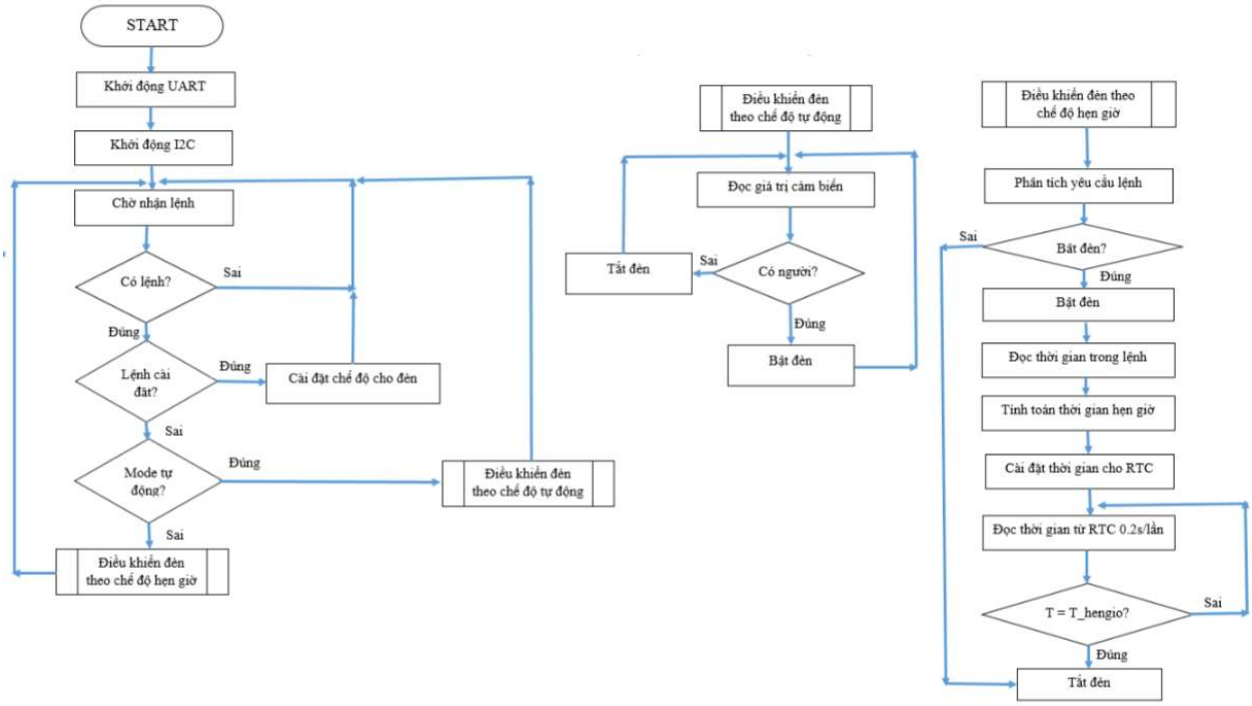

Figure 3. 2 Flowchart of Light Control Algorithm for Automatic and Timer Modes.

3.5. Circuit Diagram

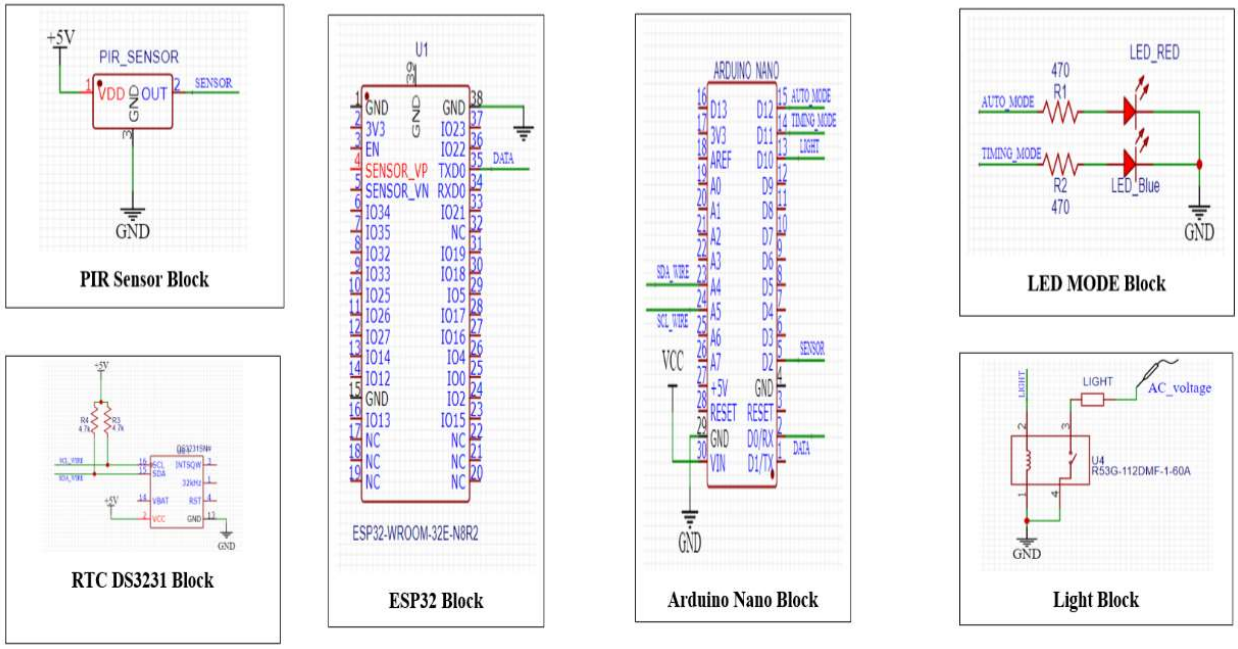

Figure 3. 3 Circuit Diagram of the System.

4. Results and Evaluation

4.1. Results

For the project "Voice-Controlled Smart Lighting System for Smart Home", the team successfully met the set requirements, specifically:

- The light operates according to the designed functions.

- The system can recognize voice commands and control the lights as requested by the user via WiFi.

- The system responds to commands with low latency (under 5 seconds).

4.2. Evaluation

Through this project, the team acquired new knowledge, specifically:

- Successfully implemented data transmission via WiFi using the ESP32 module.

- Designed and built a simple IoT system.

- Learned how to develop a voice recognition program using Python and how to program the Arduino.

- Gained understanding of data transmission between MQTT and ESP32.

5. References

[1] Speech Recognition with Python: https://reintech.io/blog/how-to-create-a-speech-recognition-system-with-python

[2] Using MQTT in Python with Paho Client: https://www.emqx.com/en/blog/how-to-use-mqtt-in-python