PRODUCT DESIGN AND MANUFACTURING

OPEN-SOURCE DOCUMENTATION

REMOTE EMERGENCY MEETING SCHEDULING DEVICE

VERSION 1

Designed by

ENG. MALEK M. ITANI

ENG. KARIM A. JIBAI

Submitted to

THE PUBLIC DOMAIN

JANUARY 2025

ABSTRACT

This report provides an in-depth exploration of the design, development, and deployment of the Remote Emergency Meeting Scheduling Device, a versatile, open-source IoT solution designed to streamline emergency communication. This device, a single-button unit housed in an ergonomic enclosure, is built to trigger urgent Google Meet scheduling via GSM-based SMS communication, circumventing potential vulnerabilities of Wi-Fi networks. Its robust integration of hardware, software, and backend technologies ensures reliability, security, and scalability, making it suitable for various industries such as corporate environments, healthcare, education, and emergency services.

The report details the iterative design process, from 3D modeling the enclosure to electronic circuit design and PCB layout, and backend server integration that manages scheduling and notifications. It also includes comprehensive instructions for manufacturing, assembly, and setup, allowing for replication and customization. By adhering to an open-source model, this solution encourages adaptability, enabling users to customize the system for diverse applications. Addressing challenges such as regulatory compliance, hardware optimization, and software efficiency, the report positions the device as a practical, scalable, and cost-effective emergency communication tool adaptable to many industries.

TABLE OF CONTENTS

1.   INTRODUCTION

1.1 Background Information

In fast-paced and high-stakes environments, the ability to rapidly respond to emergencies is essential. Whether in corporate settings, healthcare facilities, educational institutions, or industrial environments, effective communication during crises can mitigate risks, minimize downtime, and save lives. Recognizing this critical need, the Remote Emergency Meeting Scheduling Device was developed as an innovative, open-source IoT solution designed to simplify and accelerate emergency response coordination.

The device operates with a single press of a button, sending a GSM-based SMS to a backend server that schedules a Google Meet session and sends instant notifications to relevant stakeholders. Unlike traditional methods that depend on manual scheduling or Wi-Fi connectivity, this solution offers unmatched reliability by leveraging GSM communication, making it suitable even in areas with limited internet access. Additionally, the system's scalability and adaptability allow it to be applied across various sectors. For example, in the corporate world, it ensures urgent client communication; in healthcare, it can summon emergency teams; in educational institutions, it facilitates swift administrative coordination during crises.

The project's design emphasizes security, user-friendliness, and aesthetics. Its long-lasting battery supports over a year of uninterrupted operation, while its modular backend system integrates seamlessly with popular tools like Google Sheets, Twilio, and Google Calendar. Designed to be replicated, modified, and scaled, this open-source solution provides a reliable and affordable emergency communication tool adaptable to various industries, empowering organizations to manage crises effectively and efficiently.

1.2 Project Aim

The aim of this project is to create an open-source, scalable IoT device that offers a reliable, user-friendly, and secure solution for emergency meeting scheduling. The device is designed to be adaptable across various fields, such as corporate environments, healthcare, educational institutions, and emergency response scenarios. Its primary objectives include simplifying emergency communication by enabling users to schedule urgent Google Meet meetings with a single button press, ensuring that notifications and meeting links are quickly delivered to all stakeholders. The system is built with scalability and flexibility in mind, allowing it to meet the diverse needs of industries ranging from medical teams to corporate crisis management. The device is optimized for long-lasting operation, with over a year of battery life achieved through hardware and software enhancements, ensuring reliable performance with minimal maintenance. To enhance security, the device utilizes GSM technology, bypassing Wi-Fi network vulnerabilities to provide a stable and secure communication channel. Additionally, the project will be open-source, with detailed documentation, code, and design files available for users to replicate, customize, and scale the device to suit their specific requirements. The device will also feature an aesthetic and functional design, balancing professional appearance with intuitive usability. By combining robust hardware, efficient software, and a user-centric approach, this device aims to transform emergency communication, providing a cost-effective and versatile solution for a wide range of applications. The open-source nature invites collaboration and innovation, empowering the community to expand the device's potential beyond its original design. Figure 1 shows the block diagram of the project functionality.
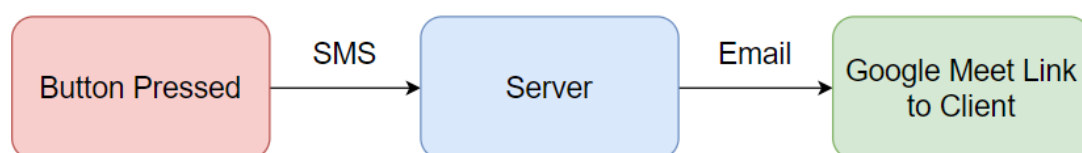

FIGURE 1: FLOW DIAGRAM OF HOW THE DEVICE SHOULD WORK

2. METHODS

2.1 Electronic Circuit Design

2.1.1 Components

This section outlines the steps taken to design the circuitry for the IoT emergency button device, detailing the components used, their roles, and the process to create a reliable and functional electronic system. The button serves as an auxiliary device, primarily responsible for sending SMS messages via GSM to initiate the meeting scheduling process. Key requirements include a battery life exceeding 365 days, reliable signal connectivity even in challenging conditions such as underground environments, and the use of rechargeable lithium-ion batteries for long-term, low-maintenance operation. The design ensures robust GSM network connectivity under tough conditions, making the device dependable in critical scenarios. Table 1 Shows the electronic components used in this project:

1. **Arduino Nano**

The Arduino Nano is a compact, microcontroller-based development board based on the ATmega328P microcontroller. It is designed to be breadboard-friendly, making it ideal for prototyping, embedded systems, and small electronics projects. The board operates on a 5V supply, powered through a mini-USB connection or an external power source. It features 14 digital input/output pins, 8 analog inputs, a 16 MHz clock speed, and a built-in USB-to-serial converter, making it suitable for and IoT (Internet of Things) system.

FIGURE 2: ARDUINO NANO

2. **SIM800L V2 EVB with Antenna**

The SIM800L V2 is an upgraded, more reliable GSM/GPRS module for SMS, voice calls, and data transfer over 2G networks. It consumes 2-10mA in idle mode and 100-200mA when sending SMS. During data transfer or calls, it can draw up to 2A, requiring a stable 5V power supply. The V2 version includes a more powerful antenna for better signal

FIGURE 3: SIM800L V2 EVB

reception, making it ideal for IoT, remote monitoring, and cellular communication projects.

3. MT3608 DC-DC Step Up Converter

The MT3608 is a compact and efficient DC-DC step-up (boost) converter that increases input voltages from 2V to 24V up to an adjustable output of 5V to 28V. It can deliver up to 2A of output current, depending on input and output conditions. Known for its high efficiency, typically around 85-90%, the MT3608 is commonly used in battery-powered devices, portable power supplies, and LED drivers. Its small size and reliability make it ideal for various electronics projects requiring voltage step-up functionality.

FIGURE 4: MT3608

4. HK4100F 3V Relay

The HK4100 is a compact 3V relay commonly used in low-power electronic projects for switching high-current loads with a low-voltage control signal. It features a 3V coil voltage and can handle up to 10A of current at 120V AC or 30V DC. This relay is widely used in automation systems, control circuits, and devices requiring safe switching of high-power components. Its small size and reliable performance make it a popular choice for microcontroller-based applications where low voltage control and high current switching are needed.

FIGURE 5: HK4100F 3V RELAY

5. SMA Female IPX Coaxial Cable (Right Angled)

The SMA Female to IPX Coaxial Cable is designed for connecting devices with SMA female connectors to those with IPX connectors, providing reliable RF signal transmission. It is commonly used in wireless communication systems, antennas, and RF modules. This cable is available in a version with a PCB-mounted SMA female connector featuring four legs for secure mounting. It offers flexible, low-loss signal transmission, making it ideal for compact electronics projects, IoT applications, and devices requiring high-frequency signal connectivity.

FIGURE 6: ANTENNA SMA TO IPX CONNECTOR

4

6. 1N4007 Diode

The 1N4007 is a general-purpose rectifier diode commonly used for rectifying AC to DC in power supply applications. It has a peak reverse voltage of 1000V and can handle a forward current of up to 1A. In addition to its use in power supplies, the 1N4007 is often used as a freewheeling diode (flyback diode) across relay coils to protect circuits from voltage spikes generated when the relay is turned off. Its durability, high voltage rating, and reliability make it ideal for protecting sensitive components in automotive, industrial, and electronic systems.

FIGURE 7: 1N4007 DIODE

7. JST XH 2.54 Male and Female Connectors

The 2-pin JST female and male connectors are commonly used in electronics for providing a secure and compact connection between two components. These connectors are widely used in power supplies, batteries, and signal transmission applications. The male connector typically has two pins that are inserted into the female connector, ensuring a reliable connection. They are designed for easy and secure installation, making them ideal for use in various projects, including robotics, IoT devices, and other compact electronic systems that require efficient, low-profile connections. For this application 3 of 3 each are required.

FIGURE 8: JST CONNECTORS

8. 104 Ceramic Capacitor

The 104 ceramic capacitor is a widely used passive component with a capacitance of 100nF (nano-farads), often utilized for decoupling, filtering, and smoothing in electronic circuits. It helps prevent accidental button presses caused by electrical noise by stabilizing the voltage and reducing signal interference. The capacitor is known for its small size, reliability, and stability across a range of voltages, making it ideal for power supply circuits, signal filtering, and noise reduction. It is commonly used in consumer electronics, automotive, and industrial devices.

FIGURE 9: 100NF CERAMIC CAPACITOR

5

9. Sanwa 30mm Red Button

The 30mm Sanwa red button is a durable and widely used push-button switch commonly found in arcade machines, gaming controllers, and electronic projects. Known for its high-quality construction, it provides reliable operation and a tactile feel when pressed. This button is often used for user input in various applications, and its large size makes it easy to press. It is designed to minimize accidental activation, ensuring precise control, and is ideal for projects requiring robust, responsive switches, such as in gaming, robotics, and DIY electronics. This model comes with a screw mechanism to fix it securely in place.

FIGURE 10: SANWA RED BUTTON

10. 18650 Battery

The 18650 battery is a widely used rechargeable lithium-ion cell known for its high energy density and long cycle life. With a nominal voltage of 3.7V and typical capacities ranging from 1800mAh to 3500mAh, it is commonly used in portable electronics, power tools, and electric vehicles. The 18650 battery is known for its reliability, compact size, and efficiency, making it ideal for applications requiring a stable power source, such as in battery packs for laptops, flashlights, and IoT projects. Three will be used for this project.

FIGURE 11: 18650 LITHIUM ION BATTERY

11. Single Cell 18650 Battery Holder

The single-cell 18650 battery holder is designed to securely hold a single 18650 lithium-ion battery, providing a reliable power source for various electronic projects. With its compact and durable design, it is ideal for powering low-voltage devices, such as an Arduino board. This holder ensures easy installation and removal of the battery, while maintaining a stable connection to power the Arduino and other small electronics. It is commonly used in DIY projects, robotics, and portable power supplies, offering an efficient solution for battery-powered

FIGURE 12: 18650 SINGLE CELL HOLDER

12. Two Cell Parallel 18650 Battery Holder

The 2-cell parallel 18650 battery holder is designed to hold two 18650 lithium-ion batteries in parallel, providing increased capacity and current output. This holder ensures a stable and reliable power source for high-current devices, such as the SIM800L module, which requires more current than a single cell can provide. By connecting the batteries in parallel, the holder doubles the available capacity while maintaining a consistent voltage of 3.7V, making it ideal for powering power-hungry components like the SIM800L, which is used in IoT and communication projects.

FIGURE 13: 18650 TWO CELL PARLLEL HOLDER

13. Male Pin Headers

Male pin headers are widely used for PCB mounting, providing a secure connection for electronic components. These headers consist of rows of metal pins, usually spaced 2.54mm apart, which are soldered onto the PCB. They are commonly used to connect components, such as sensors or modules, to the PCB, allowing for easy and removable connections. They are a standard choice for creating customizable and accessible connections in electronic designs.
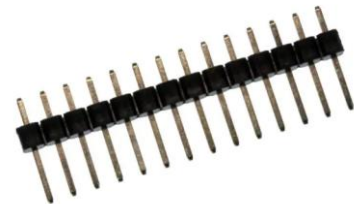
FIGURE 14: MALE PIN HEADERS

2.1.2 Circuit Functionality and Explanation

Many Iterations of the circuit were made but for this section we will only discuss the working circuit. The circuit design revolves around the Arduino as the central control unit, with careful attention to power management, signal integrity, and efficient use of components. The Arduino is powered by a single 18650 lithium-ion battery, this dedicated power source provides consistent performance and isolates the Arduino's operation from other high-current sections of the circuit.

The button is wired to one of the Arduino's GPIO pins, specifically interrupt pin 2, as the Nano has limited interrupt pins (2 and 3). A debounce capacitor is placed across the button's terminals to smooth out mechanical noise and eliminate erratic signals caused by button bounces and random noise, ensuring a clean and reliable signal is sent to the Arduino. The button serves as the primary input mechanism, triggering the circuit's operations. When

pressed, it connects the pin to ground, providing a suitable trigger for the Arduino to wake up.

A 3V relay is incorporated to manage the power supply to the SIM module. The relay coil is driven directly by the Arduino's GPIO output through a low-current connection. A flyback diode is placed across the relay coil to handle voltage spikes generated when the coil is de-energized. This protects the circuit from damage and ensures the longevity of the relay. The relay serves as an electrical switch, allowing the SIM module to remain completely disconnected from its power source until required, thereby conserving energy.

The SIM800L V2 was used for this project as the SIM800L V1, after rigorous testing, was found to be less reliable than the V2 and required 2 additional resistors to operate. The SIM800L V1 should work fine for this circuit but we decided to go with the V2 as it required less trouble shooting and simple at the expense of it being operated at 5V instead of 3.7-4.2. Also, the antenna that came with the SIM800L V1 had very poor connectivity and required a separate antenna to be bought with it whereas the V2 came with a robust antenna. The SIM module is powered by two parallel 18650 lithium-ion batteries, chosen for their ability to provide the substantial current required during GSM transmission. These batteries are connected through appropriate wiring to ensure a stable power supply, with an MT6803 boost converter employed to step up the lithium-ion batteries' voltage range of 3.5–4.2V to a stable 5V. This boosted voltage is necessary to meet the operating requirements of the SIM module and maintain its performance under varying power conditions. Additionally, bypass capacitors are strategically placed close to the SIM module's power input. These capacitors play a crucial role in mitigating voltage fluctuations that may arise from sudden current surges during transmission, effectively smoothing out the power supply and ensuring the SIM module functions reliably even under demanding conditions. More information on the sim800L can be found in reference [1].

Following the manufacturer's recommendations, it is best practice to turn on the SIM module by first connecting it to ground and then supplying Vcc. In low-voltage systems, such as those operating at 3.7V, both high-side and low-side switching can be challenging. Low-side switching, in particular, may suffer from voltage drops across the MOSFET, leading to inefficiencies and insufficient voltage for reliable operation. In these cases, a relay offers a simpler and more effective solution for controlling the SIM module's power supply, bypassing the complexities of MOSFET switching and ensuring stable power delivery. Make

sure to wire a freewheeling diode across the terminals of the relay coil to prevent a reverse current in the direction of the Arduino when the relay turns off and thus damaging it.

An external antenna is attached to the SIM module via a dedicated connector to enhance signal reception. This antenna is positioned to maximize GSM network connectivity, ensuring stable communication even in challenging environments such as underground locations. The overall layout of the circuit prioritizes efficiency and reliability, making the design robust and suitable for long-term use. Figure 13 shows the block diagram of the circuit.
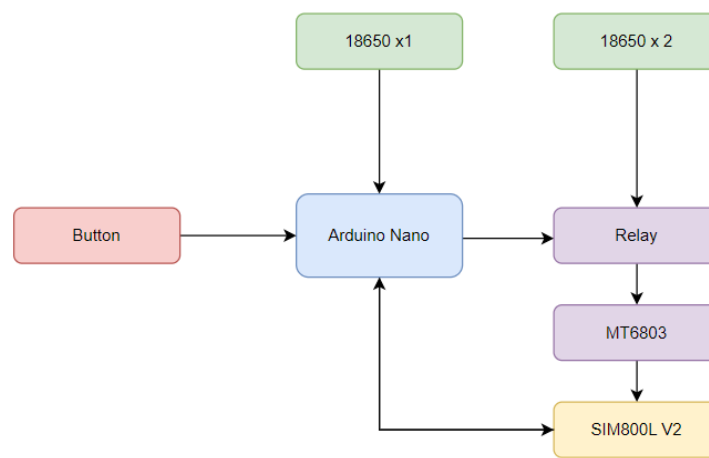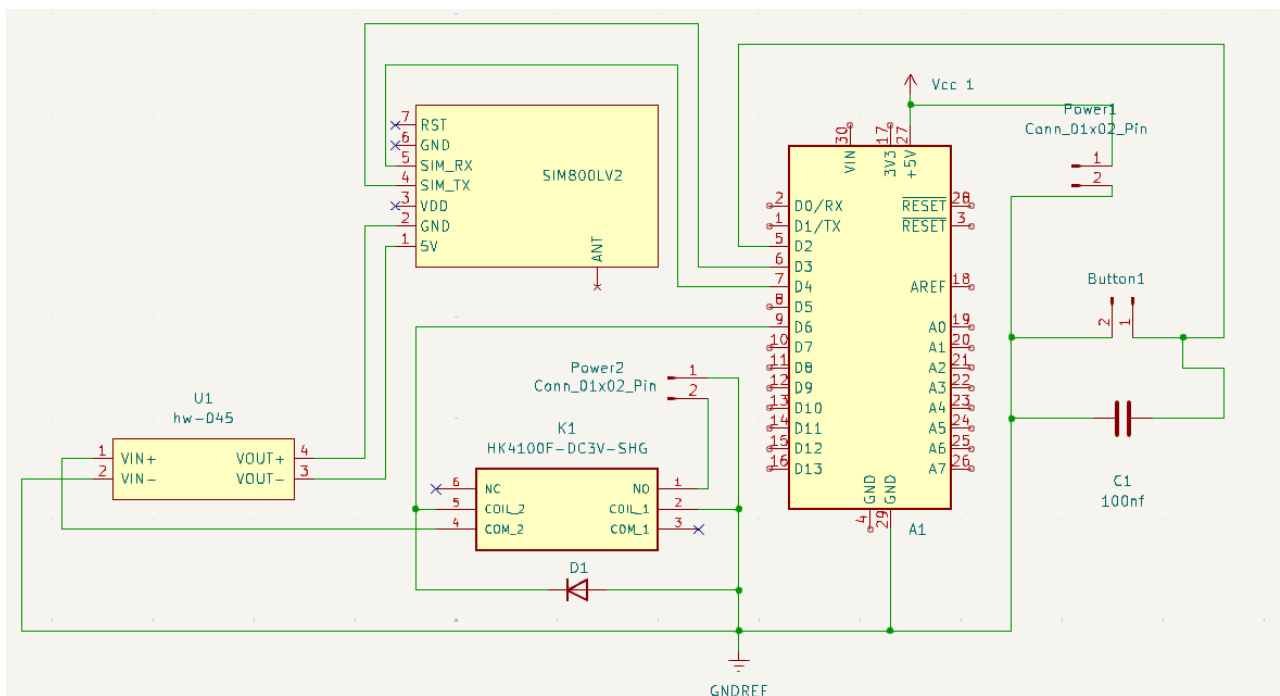

FIGURE 15: BLOCK DIAGRAM OF CIRCUIT

FIGURE 16: KICAD SCHEMATIC OF CIRCUIT.

The circuit was designed using KiCad, the schematic is shown in figure 15. This schematic can be replicated on a prototyping board for a better understanding of how the system works. When using Ki-Cad, make sure to install the necessary libraries and Files to modify the circuit. Some files are custom made. Feel free to explore KiCad libraries.

There are several important considerations regarding the circuit that should be kept in mind to ensure optimal performance and avoid potential mistakes. These notes will help increase the efficiency of the system and ensure smooth operation throughout the project.

The relay is used in this project to control the power supply to the SIM module. Below is the pinout diagram of the relay:
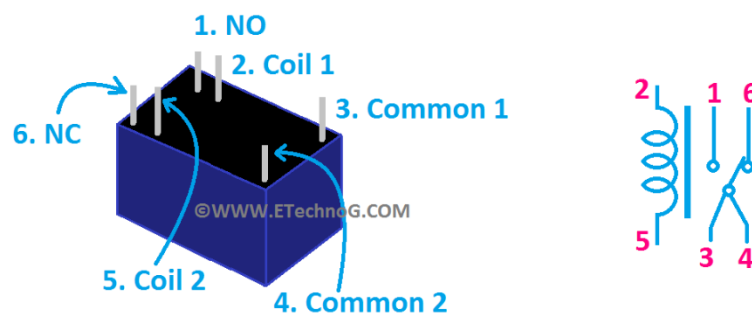

FIGURE 17: RELAY PINOUT

The relay uses the following pins for operation: Pin 2 (Control, IN) is connected to the control signal from the Arduino. Pin (Normally Open, NO) connects to the load (SIM module), while Pin 4 (Common, COM) is the shared pin used to switch the load on or off. Pin 6 (Normally Closed, NC) provides an alternative connection when the relay is in the off state (Not Connected). A freewheel diode, also known as a flyback diode, is crucial for protecting the circuit from voltage spikes caused by inductive loads, such as relays or motors. When the relay coil is de-energized, the collapsing magnetic field can generate a high voltage in the opposite direction, potentially damaging components. The freewheel diode provides a safe path for the current to dissipate, preventing damage to the circuit and ensuring reliable operation.

The Arduino Nano's TX (Transmit) and RX (Receive) pins are connected to the SIM800L module for serial communication. Specifically, Pin 3 on the Arduino Nano (RX) is used for TX of the SIM module, and Pin 4 is used for RX of the SIM. This setup allows the Arduino to send and receive data from the SIM800L, such as commands for sending SMS or making calls. If there is no communication between the SIM module and the Arduino, make sure these pins are connected properly.
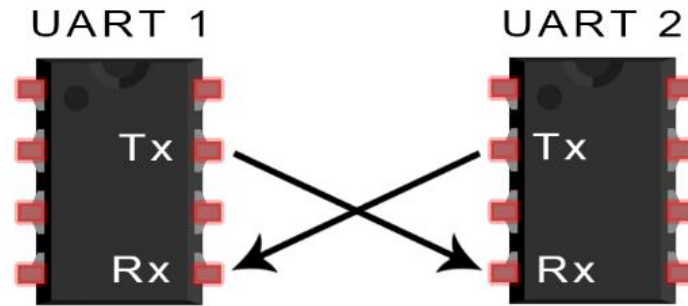
FIGURE 18: SERIAL COMMUNICATION BETWEEN ARDUINO AND SIM MODULE

For the power and button connections, 2-pin JST 2.54mm female connectors are used. These connectors provide a secure and easy-to-use interface for plugging in the power supply and the button that controls the SIM module.

There are three male JST connectors: Power 1, Power 2, and Button. These connectors are color-coded for easy identification, with the red wire indicating positive voltage (Vcc) and the black wire indicating ground (GND). It is important to ensure that all three male JST connectors are consistent, with the red and black wires on the same side of each connector.

Warning: Make sure the red and black wires are on the same side of all three connectors to avoid reversing the connections. Incorrect connections may cause damage to the circuit or components.

Power 1 is connected to a single lithium-ion battery, as it is solely responsible for powering the Arduino, which has a relatively low current requirement. Power 2, on the other hand, is dedicated to powering the SIM module. To meet the SIM module's high current demand, especially during its boot-up phase, two lithium-ion batteries are connected in parallel. This setup prevents voltage drops in the Arduino's power supply, which could otherwise cause it to reset when the SIM module draws its high inrush current during startup. Common ground is established between the 2 power supplies so that the circuit functions properly. The sim module cannot communicate with the Arduino if there is no common ground.

2.1.3 PCB Design

To integrate all components into a single compact unit, a PCB (Printed Circuit Board) was designed using KiCad. This design ensures a neat arrangement of components with minimal wiring, reducing noise and improving reliability. To keep things compact, the pcb size is set to be 8 x 8 cm with screw holes on the corners to accommodate 3mm screws. The traces are designed to avoid passing under the antenna to eliminate noise, and to keep the MT3608 as far as possible from the GSM module as possible to reduce signal interference and noise. For simplicity and cost efficiency, the circuit was implemented as a single-layer PCB, which is easier to manufacture and more affordable. The prototype PCB was manufactured in-house but can also be easily outsourced to companies like JLCPCB or PCBWay, with much higher quality and fast delivery, often within a week, for just a few dollars. Figure 19 shows the PCB Layout.
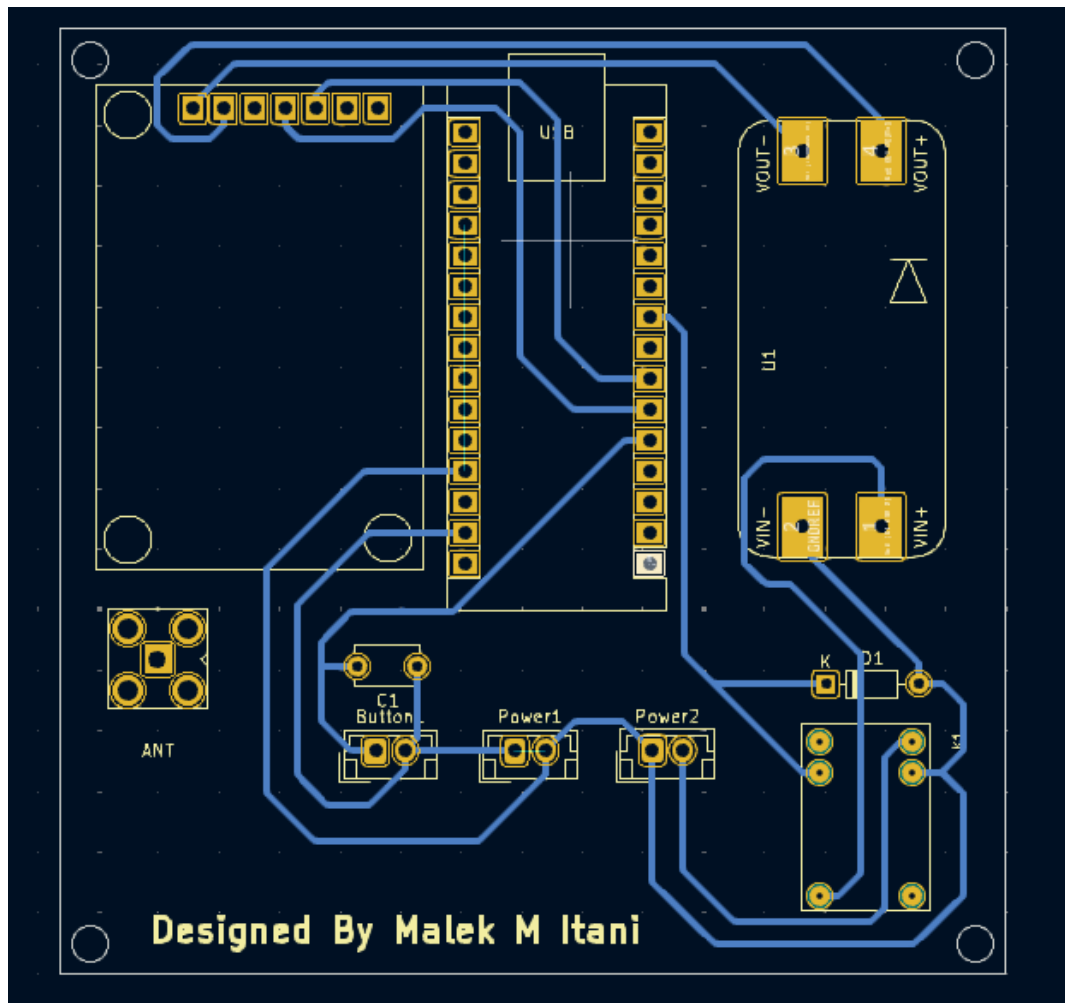

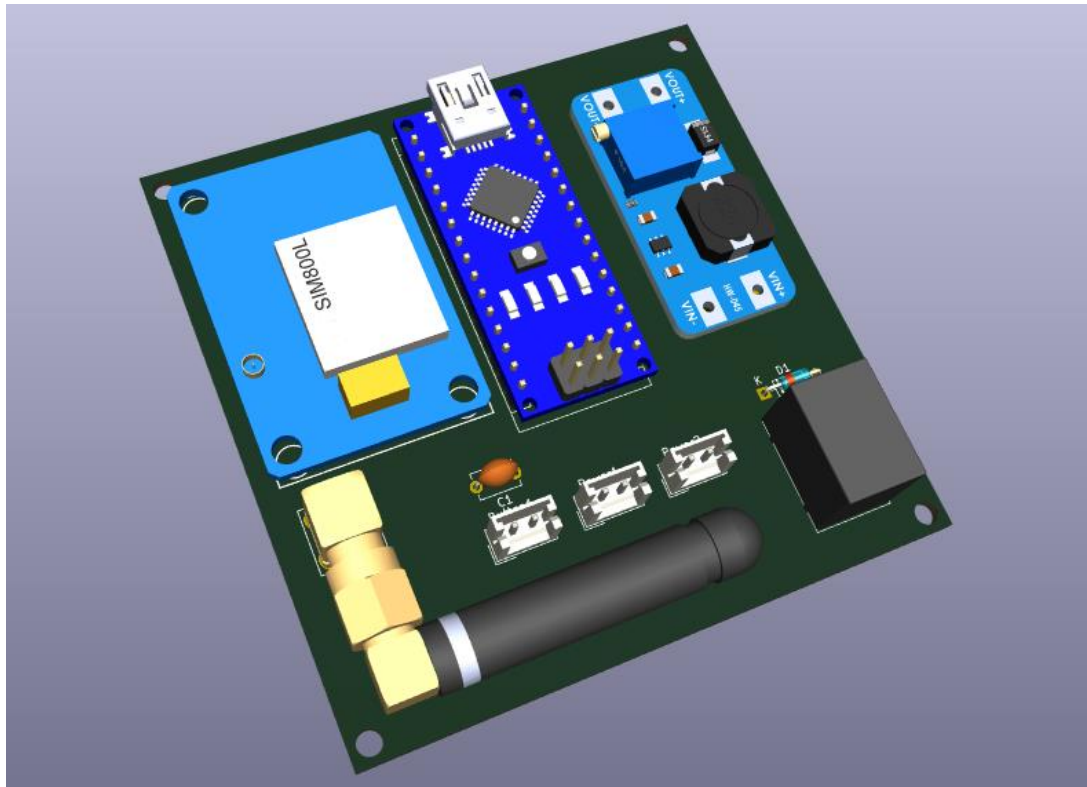FIGURE 19: PCB LAYOUT OF THE CIRCUIT

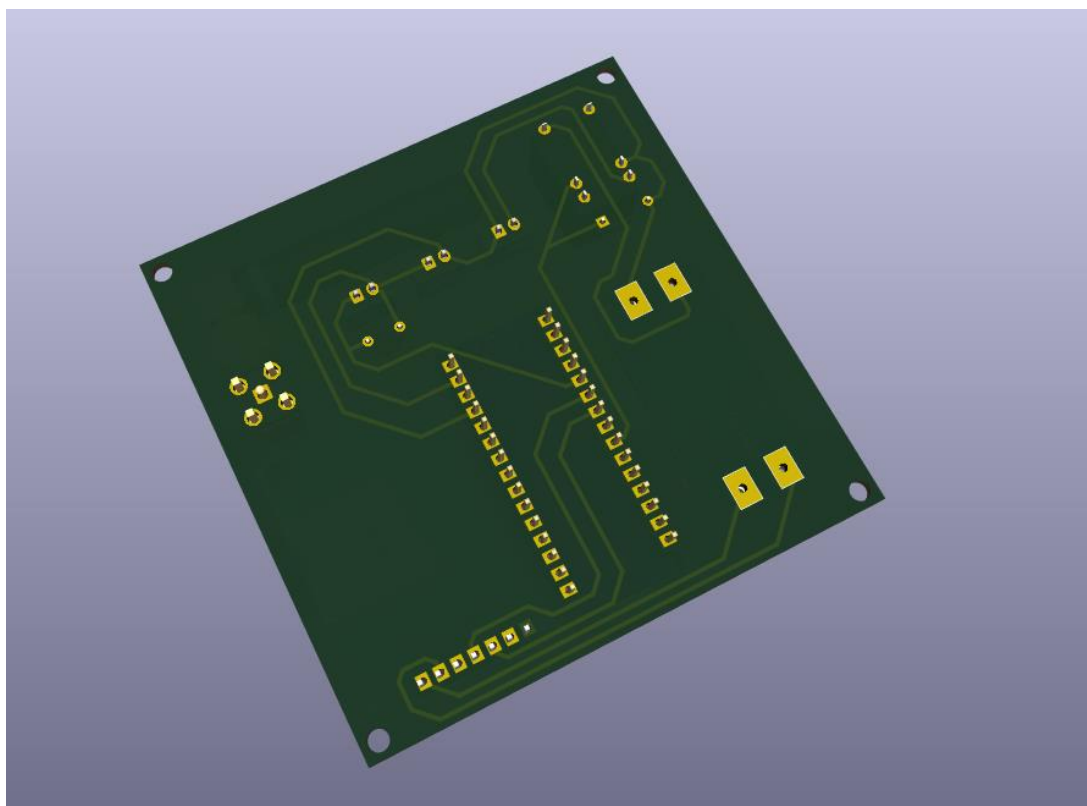FIGURE 20: 3D MODEL OF PCB FRONT VIEW

FIGURE 21: 3D MODEL OF PCB BACK VIEW

FIGURE 22: SAMPLE MOUNTING OF MT3608 ON PCB

For the MT3608, male pin headers should be soldered on the pad holes first then the module should be soldered and mounted as shown in the example board in figure 23:

The antenna mounts on the PCB and is soldered firmly in place using the SMA through hole component which is spaced and designed to keep it compact and inside the PCB boarders. This concludes the electronic design part of the project.

2.2 Arduino Programming

The Arduino code serves as the foundation for a low-power IoT device that sends an SMS notification when a button is pressed. It is designed to operate efficiently by utilizing interrupt-driven inputs and low-power sleep modes. When a button press is detected, the system wakes up from sleep, powers on the SIM800L GSM module, connects to the cellular network, sends a pre-programmed SMS, and then powers down again to conserve energy. This setup is ideal for applications such as emergency alerts or remote monitoring systems. The code integrates the SoftwareSerial library for communication with the SIM800L module and the LowPower library to implement energy-saving features. The entire code is included in Appendix A for reference. The block diagram of the code is shown in figure 24.
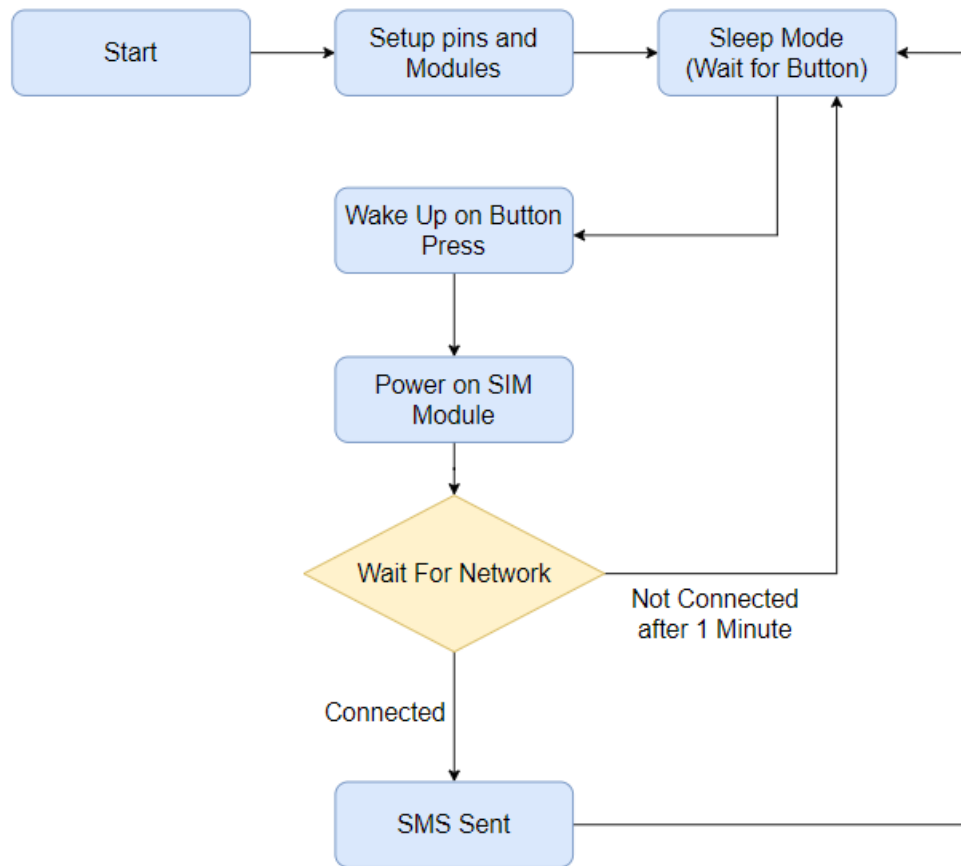
FIGURE 23: ARDUINO CODE FLOW DIAGRAM

The code begins with the inclusion of two libraries. The SoftwareSerial.h library allows serial communication with the SIM800L module using non-default pins, freeing up the Arduino's hardware UART for debugging. The LowPower.h library provides functionality to place the Arduino into deep sleep, minimizing power consumption when the device is idle.

Pin definitions are established next, with buttonPin (pin 2) assigned to the button input, relayPin (pin 6) controlling the MOSFET that powers the SIM800L, and ledPin (pin 13) for status indication via an LED. A boolean variable, buttonPressed, is used as a flag to indicate when the button is pressed. Note that for V1 of this product, the Arduino's onboard LED is used.

The SoftwareSerial object, mySerial, is instantiated to communicate with the SIM800L module via pins 3 (Rx) and 4 (Tx). This enables the Arduino to send and receive AT commands to the GSM module.

The wakeUp() function is defined as an interrupt service routine (ISR) that sets the buttonPressed flag to true whenever the button is pressed. This ISR is triggered by a falling edge on buttonPin. A falling edge is used as a trigger because the button is wired with a pullup resistor meaning that initially, its set high, and it becomes low when the button is pressed and the interrupt pin is conneted to ground

In the setup() function, all pins are configured, and initial states are established. The buttonPin is set as an input with an internal pull-up resistor to ensure stability, while relayPin and ledPin are set as outputs, starting in the LOW state to ensure the SIM800L module is initially powered off. Serial communication with the SIM800L is initialized at 9600 baud.

The main program logic resides in the loop() function. It begins by attaching an interrupt to the buttonPin to detect a button press, then places the Arduino into a deep sleep mode using LowPower.powerDown(). The device remains in this state until the ISR (Interupt service routine) is triggered by the button press. Upon waking, the interrupt is detached, and the LED briefly flashes to indicate activity.

When using an interrupt to wake up the Arduino from sleep mode, pressing the button again after the Arduino has already woken up won't trigger another interrupt because the interrupt flag has already been cleared when the Arduino is woken up. In most microcontrollers, including the Arduino, interrupts are edge-triggered, meaning they only respond to changes in the signal (e.g., a button press or release). When the Arduino wakes up, the interrupt flag is cleared, and the microcontroller no longer detects a new edge unless the button is pressed and released again, causing another transition in the signal. Simply pressing the button again while the Arduino is awake won't generate a new interrupt signal, as there is no edge (change in state) being detected at that moment.

The code then powers on the SIM800L module by setting relayPin to HIGH and waits for two seconds to allow the module to stabilize. The waitForNetwork() function is called to check the network connection status by sending the AT+CREG? command to the SIM800L repeatedly for up to 60 seconds. If the network connection is successful, the sendSMS() function is invoked to send an SMS. The SMS is sent using a series of AT commands: the mode is set to SMS text format (AT+CMGF=1), the recipient's phone number is specified, and the message content is sent, followed by a special character (Ctrl+Z) to complete the transmission.

After the SMS is sent, the SIM800L module is powered off by setting relayPin to LOW. The LED flashes again to indicate that the system is returning to sleep mode. The loop then restarts, reattaching the interrupt and placing the Arduino into deep sleep until the next button press.

The code includes additional utility functions such as updateSerial() to facilitate debugging by forwarding GSM module responses to the serial monitor. Overall, the program is designed to be efficient and modular, ensuring low power consumption while reliably handling GSM communication. The full Arduino code is provided in Appendix A for implementation and further study.

For debugging purposes, you can uncomment the Serial.print statements by removing the 2 slashes in the code to view real-time data and monitor the Arduino's behavior via the Serial Monitor. Currently, these Serial.print statements are commented out to reduce unnecessary serial communication during normal operation. By uncommenting them, you can track variable values, execution flow, and other important outputs, which can be very helpful in identifying issues or verifying that the circuit and code are functioning as expected.

2.3 Backend Code

The backend system for the IoT emergency button is a robust and modular platform designed to manage emergency communication seamlessly. It serves as the bridge between the IoT device, Google services, and end-users by processing button triggers, scheduling Google Meet sessions, and sending email notifications. The system is implemented using Flask, a lightweight Python web framework, and integrates essential APIs, including Twilio for SMS handling, Google Calendar for meeting scheduling, and Google Sheets for client data management. This modular approach ensures scalability, maintainability, and fault tolerance, making it ideal for handling real-time events efficiently. Figure X shows the flow diagram of the backend.

The database in this system is designed to manage client information and their interactions effectively. It includes two main tables: Client Data and SMS Logs. The Client Data table stores essential details about each client, such as their name, email(s), phone number, and a unique button ID, The client data is submitted via a Google Form. Multiple emails for a single client can be added, separated by commas, allowing the system to handle clients with more than one contact. The SMS Logs table records the events triggered by

button presses, such as event types (button press or battery health), along with the corresponding button ID and battery health information. These Tables can be found and monitored on Google sheets. The two tables are connected through the button ID, linking the client details from the Client Data table with the events in the SMS Logs table. This structure allows the system to identify relevant clients and their associated emails when scheduling meetings, ensuring that all the specified email addresses for a client are notified of the events recorded in the logs.
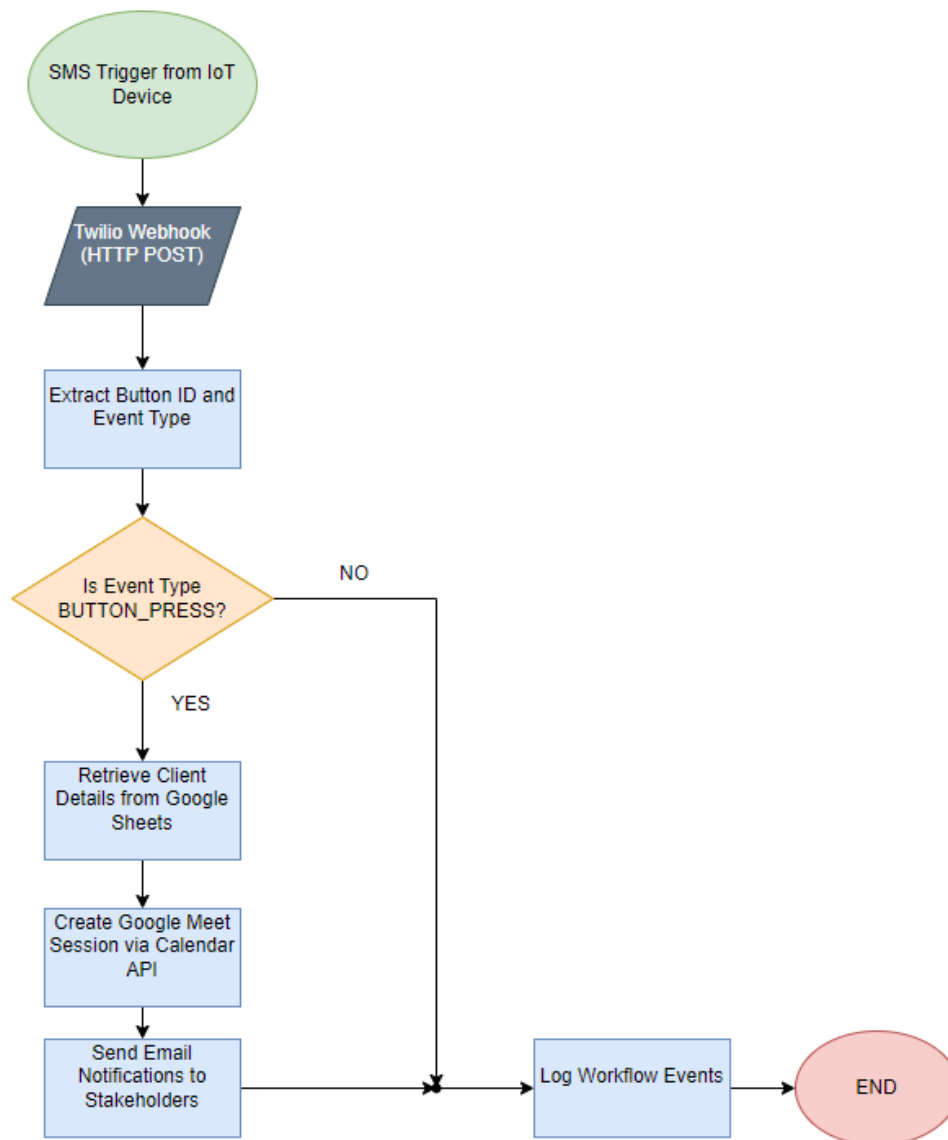

FIGURE 24: BACKEND CODE FLOW DIAGRAM

The backend workflow begins with the IoT device sending an SMS trigger to a Twilio-managed phone number. Twilio's webhook forwards this SMS as an HTTP POST request to the backend's designated endpoint, where the system extracts the relevant details such as button ID, event type, and battery health status. Depending on the event type, the backend

takes different actions. For a button press event, it retrieves client data from Google Sheets using the button ID, schedules a Google Meet session via the Google Calendar API, and sends email notifications containing meeting details to clients and team members. For a battery health event, the system logs the information without scheduling a meeting, providing a comprehensive log for monitoring purposes.

The backend's modular structure ensures streamlined operations. The `calendar_v1.py` module manages Google Calendar interactions, authenticating with Google APIs to create events that include Google Meet links, predefined durations, attendee lists, and reminders. The `sheets_v1.py` module facilitates seamless integration with Google Sheets, allowing dynamic data retrieval and updates. It supports reading client information, appending SMS logs, and managing large datasets efficiently. The `sms.py` route processes incoming SMS data from Twilio, determines the event type, and triggers appropriate actions, such as meeting scheduling or logging battery health. The `authenticator.py` service simplifies API authentication, ensuring secure and uninterrupted access to Google services.

Scalability and reliability are key features of the backend. It incorporates comprehensive error handling to manage API failures, network issues, and unexpected data formats. Monitoring tools provide insights into system performance, ensuring timely detection and resolution of anomalies. Redundancy in critical components minimizes downtime, making the system robust and efficient for real-world applications.

Setting up the backend requires enabling Google Sheets and Calendar APIs in the Google Cloud Console and securely storing API credentials in a JSON file. The application is configured through a `.env` file containing essential environment variables, such as API keys and Google Sheet IDs. The repository is easily installed by cloning the project and running the `app.py` file to launch the Flask server. This setup ensures a streamlined deployment process, enabling rapid implementation of the IoT emergency button system.

By automating repetitive tasks, such as meeting scheduling and email notifications, the backend significantly enhances the responsiveness and reliability of the emergency button system. Its integration with powerful tools and APIs ensures that the system remains adaptable and scalable, ready to handle growing demands and evolving use cases. This backend is a testament to the efficiency of modular design in creating a high-performance, user-centric solution.

Here's a step-by-step explanation of the code:

2.3.1. Modules:

`init` Module

- Purpose: Acts as an initializer, importing and exposing key modules.
- Key Elements:
  - set_calendar from calendar_v1 is renamed to calender_invite.
  - GoogleSheets is imported from sheets_v1.
  - __all__ makes calender_invite and SheetService available for external imports.

calendar_v1 Module

- Purpose: Manages Google Calendar events.
- Key Function:
  - set_calendar(invitees: list[str]):
    1. Authentication:
       - Uses GoogleAuthenticator with credentials from the config (CREDENTIAL_FILE).
       - Grants access to the Google Calendar API (CALENDER_TOKEN).
    2. Event Creation:
       - Sets an emergency meeting (summary, location, description).
       - Calculates event start_time and end_time (1-hour duration).
       - Formats invitees as attendees ([{'email': email}]).
       - Configures reminders (email: 30 mins, popup: 10 mins).
       - Adds Google Meet link via conferenceData.
    3. API Call:
       - Uses the Calendar API to insert the event into the user's primary calendar (service.events().insert).

sheets_v1 Module

- Purpose: Manages Google Sheets operations.
- Key Elements:
    - google_spreadsheet():
        - Authenticates and returns credentials and a Sheets service instance.
    - GoogleSheets Class:
        - Initializes with a SPREADSHEET_ID (from the config).
        - Exposes creds and service as properties.
        - Methods:
            - get_total_rows(sheet_name): Fetches the total number of rows in a sheet.
            - read_data(sheet_name, range_): Reads a specific range of data.
            - read_all_data(sheet_name): Dynamically reads all data from a sheet.
            - write_data(sheet_name, range_, values): Writes data to a sheet.
            - append_data(sheet_name, values): Appends data as new rows.

2.3.2. Routes

init Route

- Purpose: Imports and exposes the sms_bp Blueprint for routing.

sms Route

- Blueprint: /emergency-button
- Key Route:
    - /inbound-emergency:
        1. Handles POST requests.
        2. Extracts SMS Data:
            - Parses From, To, Body from the request form.
            - Splits the Body into button_id, event_type, and battery_health.
        3. Logs SMS Data:

- Appends the data to the SMS Logs sheet via GoogleSheets.append_data.

4. Checks Event Type:
    - If not "BATTERY_HEALTH", reads the Client Info sheet for the associated email using the button_id.
    - Sends calendar invites (calender_invite) if email(s) are found.

5. Returns a JSON response:
    - Indicates success (200) or failure (500).

2.3.3. Services

init Service

- Purpose: Creates and configures the Flask app.
- Key Elements:
    - Loads config (ProductionConfig or DevelopmentConfig based on the environment).
    - Registers the sms_bp Blueprint.

config Service

- Purpose: Manages configuration using .env variables.
- Key Classes:
    - Config: Defines shared configurations (SHEET_TOKEN, CALENDER_TOKEN, CREDENTIAL_FILE, GOOGLE_SHEET).
    - DevelopmentConfig: Enables debugging.
    - ProductionConfig: Disables debugging and testing.

2.3.4. Application

app.py

- Purpose: Entry point for the application.
- Key Elements:
    - Creates the Flask app using create_app().
    - Runs the app with debug=True on port 3000.

2.3.5. Requirements

Dependencies include:

- Flask: Web framework.
- Google APIs: Authentication and API interaction libraries (google-auth, google-api-python-client).
- dotenv: For environment variable management.

Flow Summary

1. SMS Handling:
    - Receives SMS via /inbound-emergency.
    - Logs SMS in the SMS Logs sheet.
    - If event_type is not "BATTERY_HEALTH", it sends calendar invites to associated emails.
2. Calendar Invite:
    - Sets a 1-hour meeting via Google Calendar API with reminders and a Google Meet link.
3. Google Sheets:
    - Logs and retrieves data from specific sheets dynamically.

This system enables seamless SMS logging and meeting scheduling with Google services integration.

2.4 3D Modeling

The button was designed in Fusion 360. The design process of the product evolved through several iterations, each addressing functional and aesthetic challenges. The journey began with inspiration from an AI-generated concept image shown in figure x.

FIGURE 25: AI CONCEPT IMAGE OF THE BUTTON

The initial vision focused on creating a compact, striking box with a bold design. This resulted in the first version (V1), which featured a 12x12 cm box. However, it was quickly scaled down to 7x7 cm to keep things compact. V1 included a 3D-printed red button mounted on a push button, an acrylic top engraved with "In Case of Cyber Security Emergency," and the SIM800L V1 module, chosen for its compact size. This version was powered by LiPo batteries to maintain a slim profile. V1 is shown in figure x.

FIGURE 26: BUTTON 3D MODEL V1

While promising in concept, V1 encountered significant challenges. The 7x7 cm dimensions severely limited internal space, making it difficult to fit all components. For example, the antenna could not be housed internally and had to be mounted externally, resulting in an undesirable "bomb detonator" appearance. The LiPo batteries required additional circuitry, such as a TP4056 charging module with a USB-C connector and a manual switch, which further constrained the layout and added undesirable holes in the box. The SIM800L V1 module also proved unreliable, frequently losing connectivity. Moreover, screw holes were added to mount the PCB internally, with precise clearances to ensure a secure fit, but the overall design felt too small and impractical.

To address these issues, V2 introduced significant changes. The box size was increased to 9x9x7 cm, to accommodate larger batteries (18650) and to fit all components comfortably. The switch to three 18650 batteries enhanced the circuit's battery life significantly (more than a year) and simplified maintenance. Instead of requiring internal charging circuitry, the batteries could now be easily swapped out if ever needed, ensuring long-term usability. The unreliable SIM800L V1 module was replaced with the SIM800L V2, which, although slightly larger, was far more reliable. The 3D-printed button was replaced with a professional-grade Sanwa arcade button, offering a polished red finish and a satisfying tactile response. Despite these improvements, the V2 design was bulky, plain, and did not meet the aesthetic expectations of the Red Crew. The V2 version is shown in figure x:

FIGURE 27: BUTTON 3D MODEL V2

In V3, the focus shifted to enhancing the design's visual appeal while maintaining functionality. The button's design should be a design with purpose. Drawing inspiration from arcade machines and cyberpunk aesthetics, an angled element to the button was added. Initially, all edges were filleted to soften the design, but this approach gave the box an unattractive, overly rounded appearance. V3 is shown in figure 28.

FIGURE 28: BUTTON 3D MODEL V3

In V4, a triangular hood angled above the button was introduced. This design element was discovered accidentally through a glitch during modeling but was retained for its unique look. The fillets were replaced with chamfers and sharp, triangular features, achieving a sleek, modern, and edgy style. V4 is shown in figure 29.

FIGURE 29: BUTTON 3D MODEL V4

Further refinements were introduced in V5, maintaining the chamfers and triangular aesthetics while optimizing the hood's angle for balance and visual appeal. However, the matte 3D-printed finish of earlier iterations still felt underwhelming. To address this, V5 introduced a hybrid approach: black acrylic covers mounted on a 3D-printed skeleton. This combination elevated the product's look, making it feel more luxurious and professional.

FIGURE 30: BUTTON 3D MODEL V5

The acrylic covers give the design a polished and glossy look. They are laser cut on 2.5mm back acrylic. 2mm also works fine. The covers are shown in figure 31.
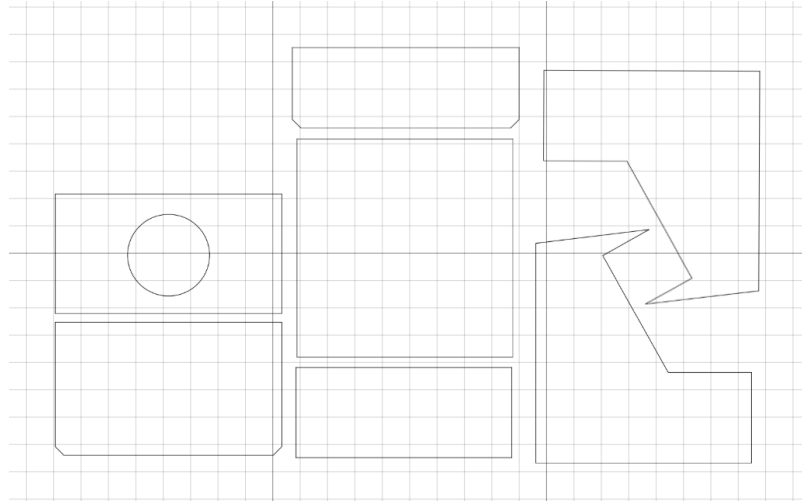
FIGURE 31: ACRYLLIC COVER LASER CUT DESIGN

The evolution from V1 to V5 reflects a balance between functionality and design. Starting with a compact but impractical box, the product matured into a robust, visually appealing device that meets technical requirements while embodying an edgy and modern cyber design.

The full product assembly is done on fusion 360 showcasing the internals and assembly of the product. The PCB is mounted with 4 M3 screws to the base. Underneath the PCB lie the 3 lithium batteries with cables extending from the battery cases to the PCB. The
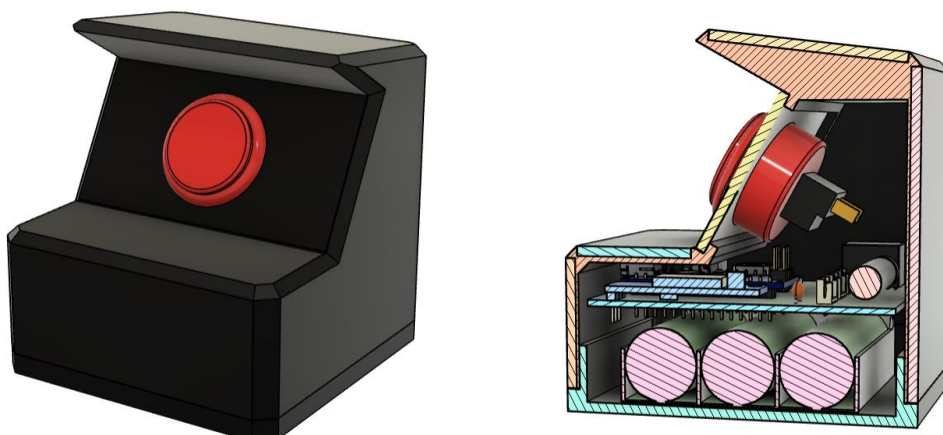
electronics also fit with enough clearance to avoid touching the box cover. This is shown in figure. X
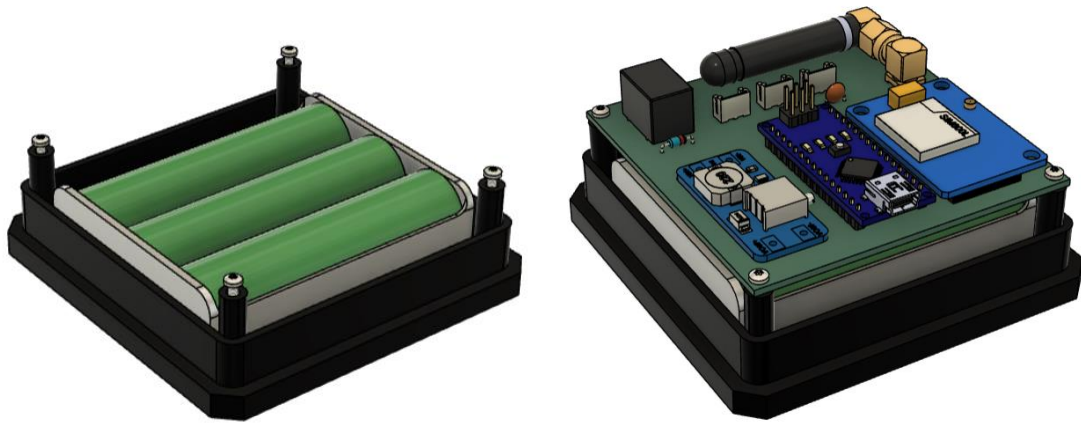
FIGURE 32: ASSEMBLED 3D MODEL WITH CROSS SECTION

2.5 Manufacturing and Setup Instructions

The manufacturing process and setup of the IoT emergency button device are designed to be scalable and efficient, ensuring high-quality output while keeping costs and complexity manageable. Two versions of the enclosure are provided: one fully 3D-printed and the other with a combination of 3D-printed components and laser-cut acrylic covers for a premium finish.

1. 3D Printing and Laser Cutting

The enclosure and internal mounts are printed using PLA with a layer height of 0.2mm for precision. A 20% infill is recommended for structural stability while keeping weight manageable. The enclosure consists of the main box, and the main base. internal mounts. The main box has 2 versions. The fully 3D printed version and the acrylic version. If the acrylic version is printed, make sure to laser cut the included covers in 2.5mm black acrylic. Post-printing, remove any supports and sand the edges for a clean fit. Ensure that all screw holes and slots align as per the design specifications. Any local 3d printing service can print this with no problem. For Aesthetics try to print the button in black.

2.   Arduino Hardware Modifications

To maximize power efficiency of the Arduino, some hardware modifications need to be done to achieve the 1- year battery life goal. Without hardware or software modifications the Nano measured to draw about 18mA of current. Note that the circuit will stay on for a long time. This configuration gives us:

$$Capacity = Amps(A) * Hours(H)$$

$$Hours = \frac{Capacity}{A} = \frac{2500mah}{18ma} = 138.8 \ hours = 5.75 \ days$$

Sleep mode alone brings down the Arduino's consumption to around 8mA (measured). If we are using a 2500mah Battery the lifetime will be:

$$Hours = \frac{Capacity}{A} = \frac{2500mah}{8ma} = 312.5 \ hours = 13.02 \ days$$

The always on power onboard LED draws around 5 mA. Removing it as seen in figure x will give us:

$$Hours = \frac{Capacity}{A} = \frac{2500mah}{2ma} = 416 \ hours = 17.33 \ days$$

The last modification to be done is to remove the onboard voltage regulator. As the Nano has a ATMega328p microcontroller, it can run on 3.3 volts comfortably. It should have no problem running on 3.7 volts or higher. The voltage regulator draws unnecessary current so it must be removed. The results of this final modification gave outstanding results which will be seen in the results section. The modifications are seen in figures X and X. The TX LED might also be on if serial communication is taking place so its best to remove it as well. They can be removed with needle nose pliers or wire cutters. They can also be desoldered.
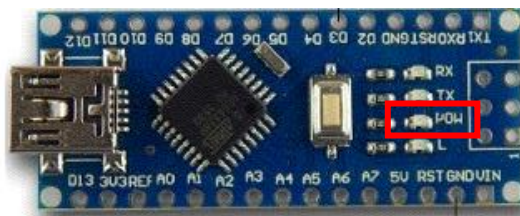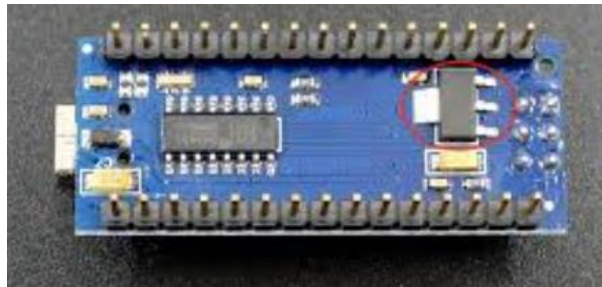

FIGURE 33: POWER LED TO BE REMOVED

FIGURE 34: ONBOARD VOLTAGE REGULATOR TO BE REMOVED

3. PCB Manufacturing and Soldering

The PCB layout is created using KiCad, adhering to an 8x8 cm size. Mounting holes for M3 screws are included for secure attachment. Upload the Gerber files provided to a service like JLCPCB or PCBWay for production. Even a local PCB manufacturer can do this as it's a simple single layer board. Make the PCB with silkscreen, solder mask, and tinned pads for highest quality.

Once the PCBs arrive, solder the components onto the board, starting with the MT3608 boost converter using male pin headers as shown in figure 23. If the header comes with many heads break 4 individual pins to fit them in the slot for the mt3608 and solder them carefully. Follow this with other components like the relay, diode, Arduino, and JST connectors.

The Arduino Nano is powered by a single 18650 lithium-ion battery. Its connector is labeled as power 1, The two-cell holder is wired in parallel for the SIM800L module, ensuring sufficient current delivery. Its connector label on the PCB is known as power 2. Solder JST connectors for the batteries, ensuring the correct polarity by verifying the color-coded wires and PCB layout. Mount the Sanwa button to the acrylic button cover or 3d printed button cover using the orange screw that comes with the button. The button polarity does not matter as the button is simply a switch. The button JST connector is labeled as "Button" on the silkscreen.

Solder the relay and its accompanying 1N4007 diode. Position the diode across the relay terminals to protect the circuit from voltage spikes. Make sure to check the polarity of the diode because reversing the diode polarity will lead to instabilities. The silkscreen should show the correct orientation. Mount the Sanwa button to the acrylic or 3d printed button cover using the included orange screw. Solder wires to the button, and battery cases

measuring and trimming to avoid excess length. Use heat shrink tubing to secure connections and prevent short circuits. Do this for all connections.

4. Arduino Program Setup

This section provides detailed instructions to set up the Arduino part of the IoT emergency button system, ensuring it is ready for deployment. Follow these steps to configure the device, customize it for specific users, and debug the system if necessary.

Using Arduino IDE, Upload "Emergency_Meeting_Scheduling_Device_Arduino_Code_Working" sketch BEFORE connecting the batteries via its micro-USB port. You can upload the code while the batteries are installed but it is safer to install them later to avoid risking damaging your hardware.

Open the Arduino IDE and navigate to the SendSMS() function in the code.

```
void sendSMS() {
  //Serial.println("Sending SMS...");
  mySerial.println("AT+CMGF=1"); // Set module to SMS text mode
  updateSerial();

  mySerial.println("AT+CMGS=\"+000000000\""); // Replace with your phone number
  updateSerial();

  mySerial.print("Message from Arduino: Button was pressed!"); //  //replace mess
  // event type is either a button press or a check battery. the check battery is
  mySerial.write(26); // Ctrl+Z to send the SMS
  //Serial.println("SMS Sent!");
}
```

FIGURE 35: CODE TO BE MODIFIED

Replace the placeholder phone number +000000000 with the Twilio Server phone number. You can use any number if using for SMS purposes only. This number that is inserted is the number of the recipient that will receive the SMS.

mySerial.println("AT+CMGS=\"+000000000\""); // change to recipient number.

Update the Button ID variable to assign a unique identifier for the device, useful in systems with multiple buttons.

String ButtonID = "001";

Replace "Message from Arduino: Button was pressed!" with "001,BUTTON_PRESS,100%" when using with server to send emails. Do not put spaces in this message copy and paste it as is. Change Client ID if there are more than 1 buttons. Any message can be sent if not using the device it with the server.

mySerial.print("Message from Arduino: Button was pressed!");

mySerial.print("001,BUTTON_PRESS,100%");

The event type is either a button press, or a check battery. the check battery event is for future improvements not included in this code, where the Arduino can monitor the battery voltage. another code is provided for monitoring battery health.

To enable debugging, uncomment the Serial.print lines in the code to enable debugging through the Arduino Serial Monitor. This allows real-time feedback on operations such as network connectivity, SMS sending, and button presses. They are by default disabled to avoid setting the Arduino's TX light on constantly and to make the code run faster. Use the Serial Monitor in the Arduino IDE to observe system behavior during testing.

5. Assembly

Place the fully charged 18650 batteries into their holders (should read 4.2 volts at maximum charge) and connect them to the JST connectors. DOUBLE-CHECK THE BATTERY POLARITY BEFORE POWERING THE CIRCUIT. In this version, there is no reverse polarity protection in this version so please be careful when plugging the batteries. Then, insert the battery holders in the base of the "Main Base" part, then secure the PCB to the base of the enclosure using four M3x10 screws. Ensure all components have adequate clearance. Attach the button cover to the main box through the dedicated square hole and glue it in place with super glue, ensuring it does not interfere with the internal components. Attach the box housing by simply sliding it in with the friction-fit design. Such that there is no play and the base with the batteries won't slide out easily even with heavy shaking. Ensure all external components, such as the SMA antenna, are securely mounted and soldered in place. The assembly is shown as an animation provided in the README file in the repository.

Use an active SIM card with SMS capabilities. Insert the SIM card into the SIM800L module's slot, ensuring the correct orientation as indicated on the SIM800LV1 module as shown in figure 36. The same applies for the SIM800L V2. The sim card slot is on the side of the PCB making it accessible even if soldered in place. Any 2G Micro SIM card will work perfectly. If the sim card is too small use a SIM card adapter for it to fit on the module. The proper way to insert the SIM card is typically engraved on the surface of the SIM socket.
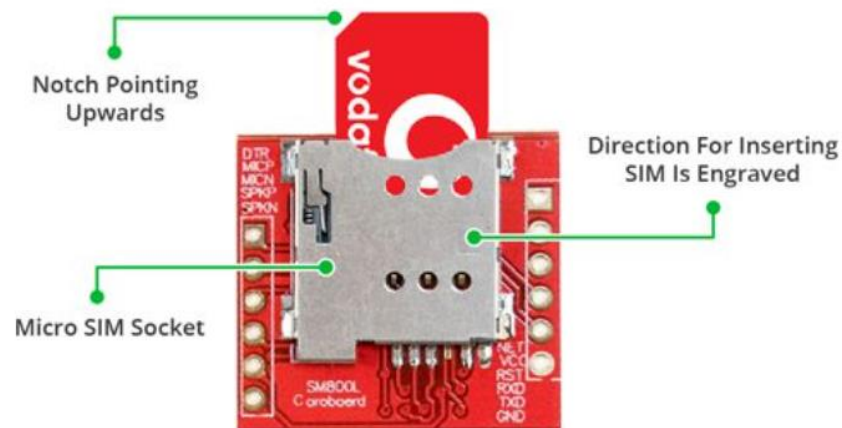
FIGURE 36: SIM CARD INSERTION ORIENTATION.

To Finalize the SIM Connections, screw in the external antenna to the board to enhance GSM signal reception, especially in areas with weak signals. Make sure the antenna is mounted and soldered in properly to the SMA connector and that the antenna cable is plugged into the SIM800L IPX slot and tucked inside the edges of the PCB.

3. RESULTS AND DISCUSSION

The results section presents the outcomes of the design, development, and testing processes for the IoT emergency button system. This device was evaluated based on its functionality, reliability, and adherence to project objectives. The results encompass all critical aspects, including the electronic circuitry, software implementation, 3D-printed housing, and overall system performance under various test conditions. This section highlights the device's ability to meet the outlined requirements, such as long battery life, seamless GSM-based communication, and robust design. Additionally, challenges encountered during the development phase are addressed, along with their resolutions, to provide a comprehensive assessment of the project's success.

FIGURE 37: FINAL BUTTON 3D PRINTED PROTOTYPE DESIGN
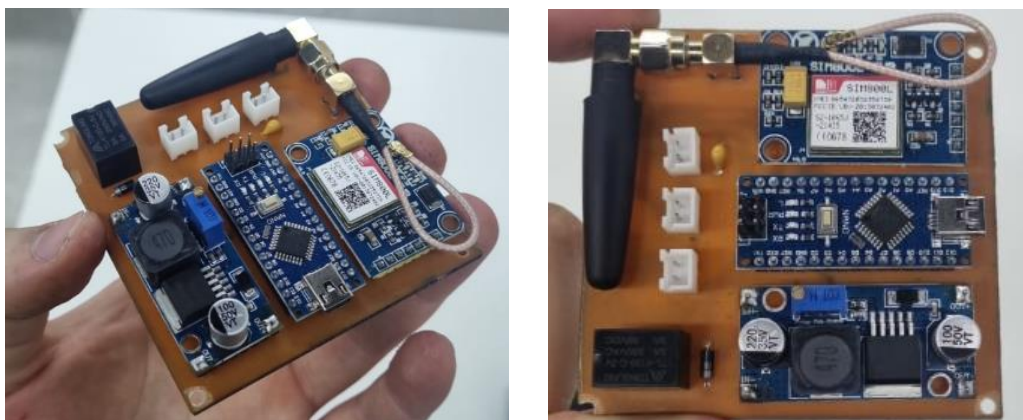
FIGURE 38: ASSEMBLED PCB

Note that the boost module in figure 38 is the XL6009 Boost module but was later swapped to the MT3608 because according to the data sheet of the XL6009, it only boosts voltages starting from 5 volts and above even though it did work fine in this application but it is better to stick to the manufacturers recommendations to prevent future issues. The MT3608 is better suited for such purposes as indicated in reference [4] as it supplies up to 2 amps in theory and works well with 3.7-volt batteries. It is advisable to add a 470 uF capacitor at the VOUT terminals of this module. To minimize noise. The included PCB schematic is updated to accommodate the MT3608.
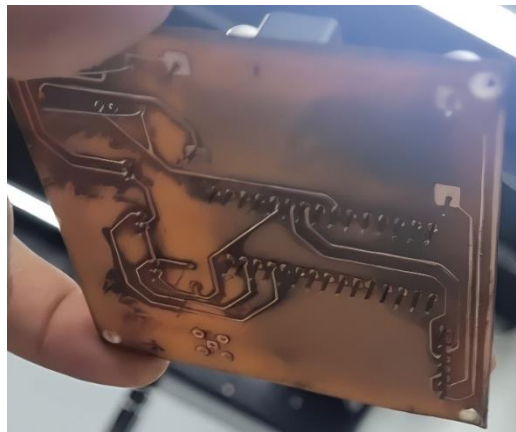
FIGURE 39: DIY PCB TRACES

Due to time constraints, the PCB was manufactured in house, thus, the result is a crude looking PCB that works great. It's advisable to manufacture the PCB in a professional environment by using service providers like PCB way.
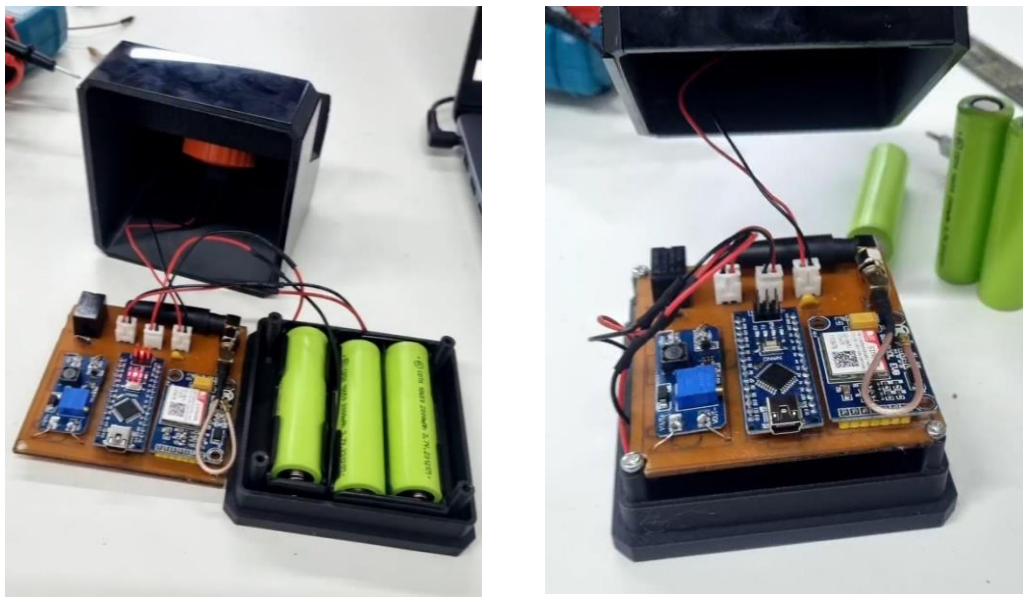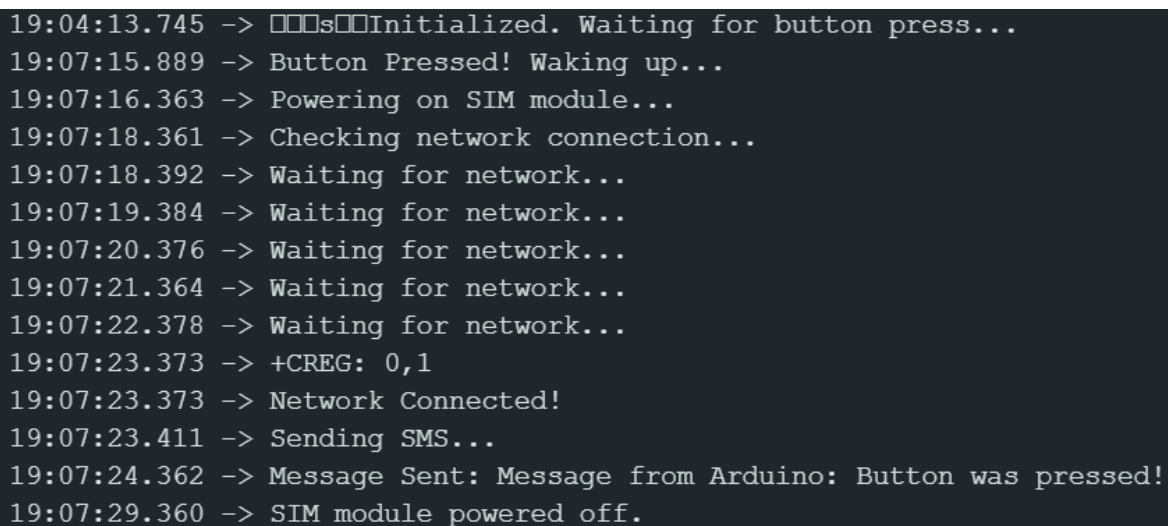

FIGURE 40: FINAL ASSEMBLY OF THE DEVICE

Once all connected and assembled, all is left is to press the button. Once the button is pressed, the Arduino wakes up from sleep mode, the relay coil is energized, the Arduino onboard LED flashes, and an audible click should be heard from the relay and the sim module turns on. The SIM module should start to blink once every 1 seconds for about 10 seconds before connecting to the network. Once it connects to the network, the SIM module should blink once every 3 seconds. More information on how the sim module should behave is shown in reference [x]. Then, a few seconds will pass waiting for the module to stabilize which then leads to the message will be sent. Once the message is sent, the onboard LED will flash for half a second, the SIM module will stay on for 2 seconds to ensure the proper delivery of the message and then it will turn off. The relay will tick again de-energizing the coil, switching the relay contact, and turning off the module. The functioning of the circuit can be viewed in the Arduino IDE Serial monitor by uncommenting the Serial.print functions. The working result is shown in figure 41

```
19:04:13.745 -> □□□s□□Initialized. Waiting for button press...
19:07:15.889 -> Button Pressed! Waking up...
19:07:16.363 -> Powering on SIM module...
19:07:18.361 -> Checking network connection...
19:07:18.392 -> Waiting for network...
19:07:19.384 -> Waiting for network...
19:07:20.376 -> Waiting for network...
19:07:21.364 -> Waiting for network...
19:07:22.378 -> Waiting for network...
19:07:23.373 -> +CREG: 0,1
19:07:23.373 -> Network Connected!
19:07:23.411 -> Sending SMS...
19:07:24.362 -> Message Sent: Message from Arduino: Button was pressed!
19:07:29.360 -> SIM module powered off.
```

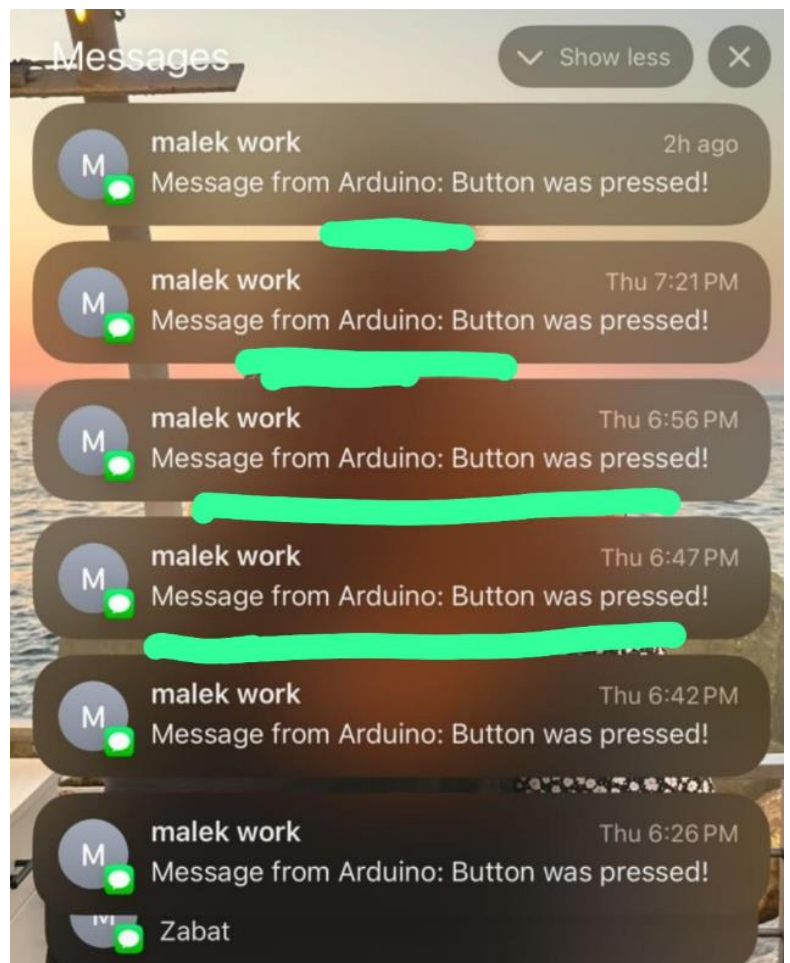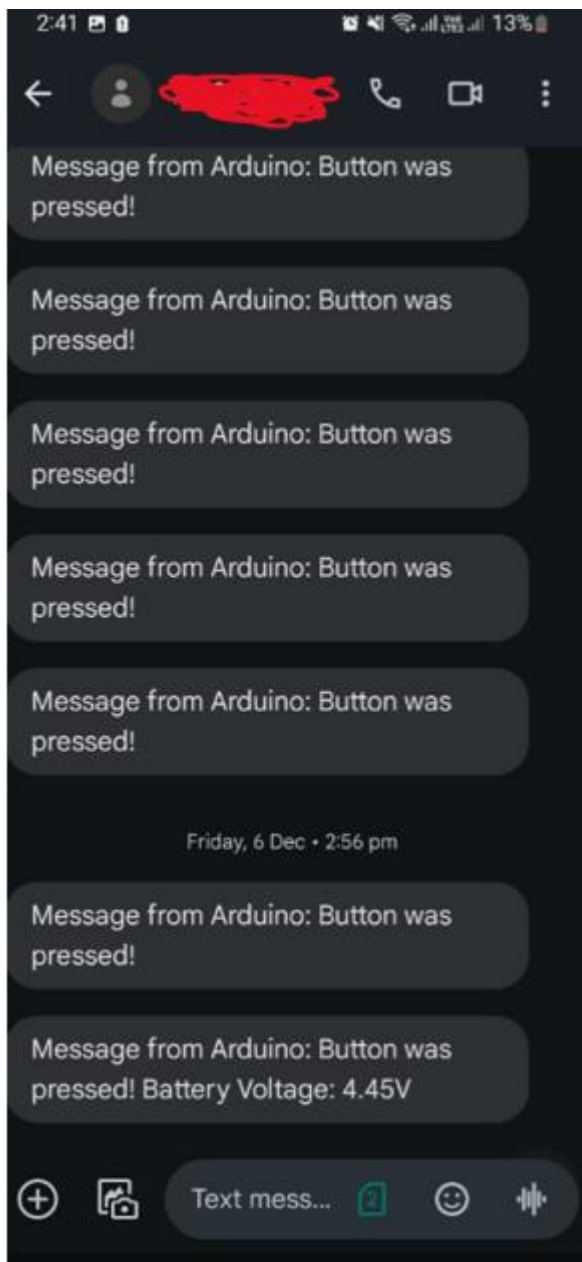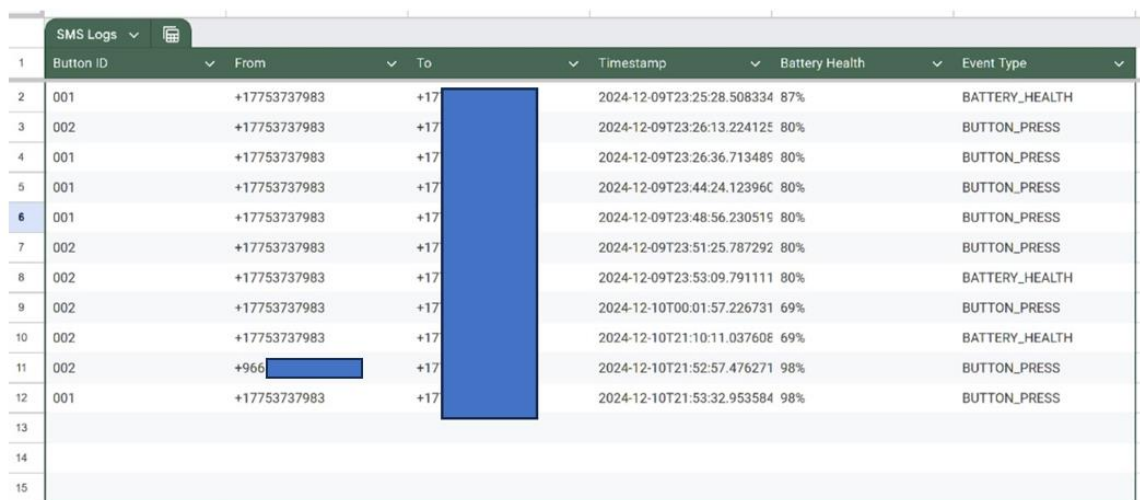FIGURE 41: SERIAL MONITOR SHOWING THE SENDING OF THE SMS AND THE
WORKING OF THE DEVICE

FIGURE 42: RIGOROUS TESTING ON MULTIPLE RECIPIENTS WHO RECEIVED
MESSAGES FROM THE ARDUINO WHEN THE BUTTON WAS PRESSED

The button was tested hundreds of times to make sure it works properly. A trouble
shooting guide is provided in the next section. The battery monitoring function was a test
later to be discussed.

Now that the button successfully sends SMS messages, it can be hooked to the server. Practically, the button is just a message sending device. Any phone can interact with the server if the right SMS is sent to the right number. Simply replace the "Message from Arduino: Button was pressed!" with the button id, event type, and battery health percentage. And make the recipient of the message the server recipient phone number in Twilio. This enables the device to communicate with the server and make it work properly. The event type in this project version is limited to BUTTON_PRESS but in the future can be modified to monitor the battery percentage with the BATTEY_HEALTH event. Figure 43 shows the SMS logs linked to the google sheet document showcasing all the events that took place.

FIGURE 43: SMS LOGS OF ALL MESSAGES RECEIVED TO THE SERVER AND THE EVENT TYPES

Different numbers were tested from different countries. They all worked except attempting to send the number from Lebanon as sending SMS messages from Lebanon to international numbers or American numbers do not work, which is unfortunate. However other numbers from different countries such as Twilio's virtual number and a Saudi Arabian number worked fine. It is preferred to register and verify in Twilio, a local number for the server from the country that the button will be deployed in.

When the event type is received as BUTTON_PRESS, the code will check the button ID and compare it with the client ID retrieved from the Client data table shown in figure 44 which contains the client's data such as emails, name, and phone number. This data is initially filled out by the client or the company using a google forms document shown in figure x which when submitted, automatically saves the information in the client data table, each client has a unique button ID, list of emails, name, and telephone number, which is used by the server to know to which client to send the emails to.

FIGURE 44: CLIENT DATA TABLE

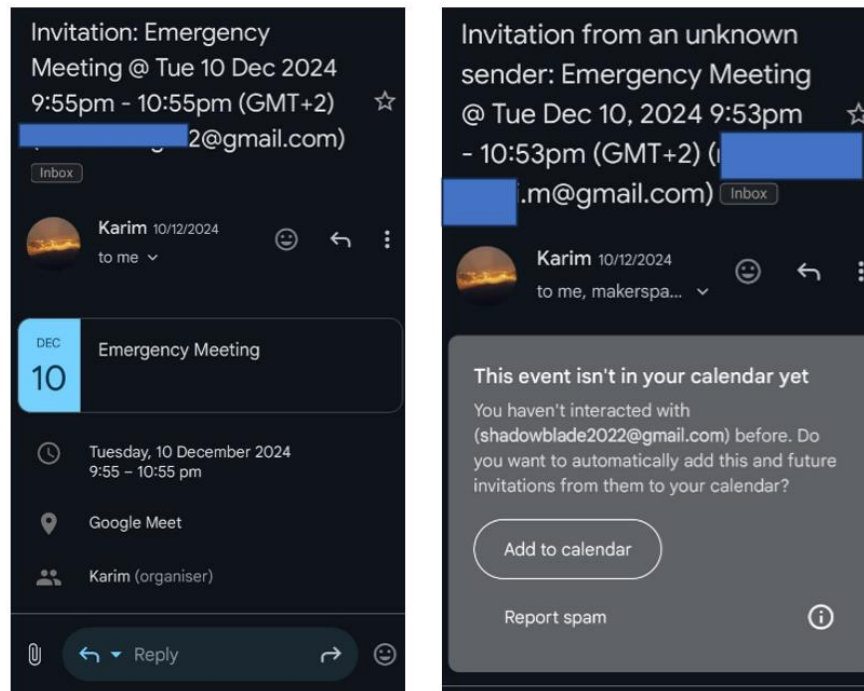FIGURE 45: GOOGLE FORM TO FILL IN CLIENT DATA

FIGURE 46: RECEIVED INSTANT MEETING EMAILS VIA GOOGLE CALENDAR

During Testing, 2 SMS messages were sent at different times each with a different button ID which is why 2 emails were received at different intervals. The same Button ID can handle multiple emails. This can be done by filling out in the google form multiple emails separated by a comma. This was done in other tests successfully, video links showing this will be shown in the repository.

Finally, one of the goals was to make a device with a battery that can last up to a year. The Arduino has its own independent power source. It is important to calculate the Arduino's battery span and the SIM800L's battery life as they are 2 different power sources. By implementing sleep mode to reduce power consumption in standby mode, switching off the SIM module completely when not using it, and modifying the Arduino's hardware, the result is a device that when in standby mode consume only 80 micro-Amps. Assuming the button is pressed once per day, the Arduino wakes up from sleep mode and draws about 12 mA for 1 minute to trigger the circuitry. Assuming the button is pressed once per day and the Arduino itself has a single 2500mAh battery. The result for the Arduino:

$$Standby\ current = 80\mu A = 0.08mA$$

$$Button\ Press\ current = 12mA$$

$$1 \ minute = \frac{1}{60} \ hours = 0.0167 \frac{hours}{day}$$

Energy consumption during button press (1 minute per day):

$$\frac{milli \ Amp \ hours}{day} = 12ma \times 0.0167 \frac{hours}{day} = 0.2004 \frac{mAh}{day}$$

Energy consumption during Standby Mode (approximately 24 hours per day):

$$\frac{milli \ Amp \ hours}{day} = 0.08ma \times 24 \frac{hours}{day} = 1.92 \frac{mAh}{day}$$

Total Arduino Consumption per Day:

$$Total \ Daily \ Consumption = Standby + Button \ Press = 1.92 + 0.2004 = \frac{2.12mah}{day}$$

$$Arduino \ Battery \ Life = \frac{2500mAh}{2.12 \frac{mAh}{day}} = 1179.24 \ days = 3.23 \ years$$

As for the SIM800L, its switched completely off by the relay using the normally open contact most of the day. The sim module turns on for 1 minute every day assuming a daily single button press. When on, the SIM800L draws about 200mAh for 1 minute then shuts off. The SIM800L has its own dedicated power supply (2x18650 2500mAh batteries in parallel totaling 5000mAh of capacity).

$$SIM800L \ Standby \ current \ (off) = 0mA$$

$$Button \ Press \ current = 200mA$$

$$1 \ minute = \frac{1}{60} \ hours = 0.0167 \frac{hours}{day}$$

Energy consumption during button press (1 minute per day):

$$\frac{milli \ Amp \ hours}{day} = 200mA \times 0.0167 \frac{hours}{day} = 3.34 \frac{mAh}{day}$$

Energy consumption during Standby Mode (approximately 24 hours per day):

$$\frac{milli \ Amp \ hours}{day} = 0mA \times 24 \frac{hours}{day} = 0 \frac{mAh}{day}$$

Total SIM800L Consumption per Day:

$$Total\ Daily\ Consumption = Standby + Button\ Press = 0 + 3.34 = \frac{3.34mAh}{day}$$

$$SIM800L\ Battery\ Life = \frac{5000mAh}{3.34\frac{mAh}{day}} = 1497days = 4.1\ years$$

These results exceeded completely the 1-year goal with room to spare. However, it is important to regularly check and maintain the device for best performance, and to use high quality batteries and make sure they're fully charged before using. Practically speaking, the battery life is less than the theoretical calculations but even with a 50% margin of error (which is unlikely) the lifespan should meet our requirements. Figure 47 shows the Arduino's outstanding 80 microamp current draw at 3.7 volts in figure x. The power supply couldn't even detect the current draw.
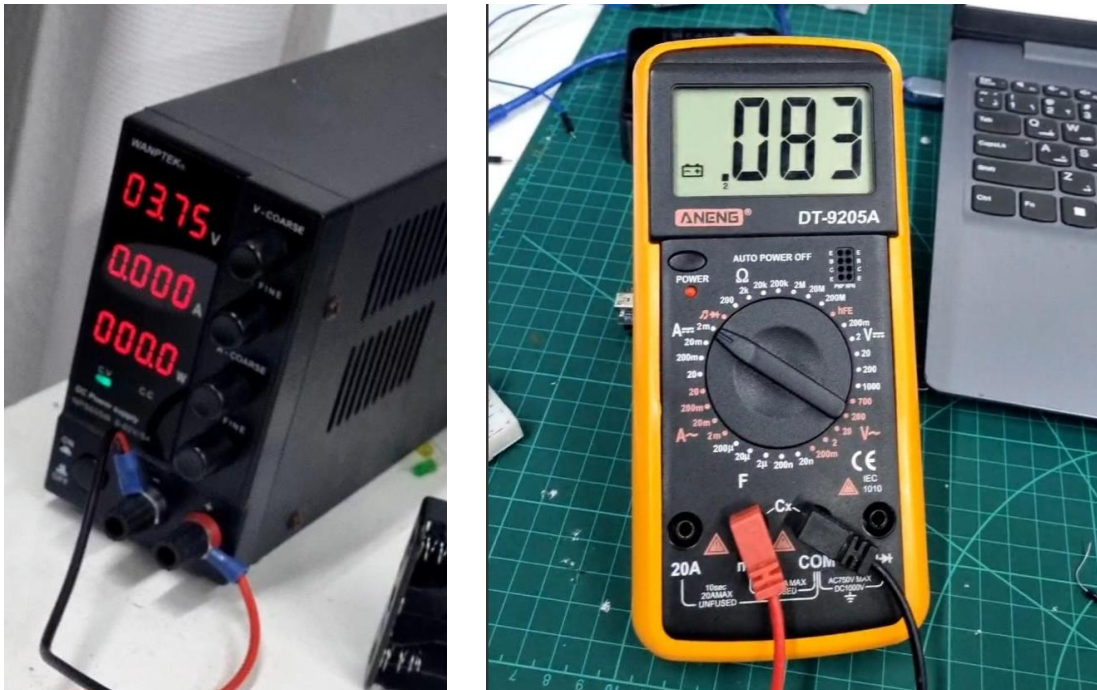

FIGURE 47: ARDUINO POWER CONSUMPTION DURING STANDBY MODE

3.2   Troubleshooting Guide

This troubleshooting guide provides a comprehensive overview of common hardware issues encountered during the development and operation of the IoT emergency button device, along with detailed solutions. By addressing problems systematically, users can ensure the device operates reliably and meets design expectations.

The SIM module is often the primary source of challenges. If the module fails to connect to the network, it may respond to AT commands but not register on the network. This could be due to an inactive SIM card, unsupported GSM bands, antenna issues, network access denial (discussed later), or poor signal strength. To resolve this, ensure the SIM card is active and test it in a mobile phone to verify network connectivity. Replace or reposition the antenna to enhance reception, and verify that the SIM module complies with local regulations. Check the SIM card if it is inserted in the correct orientation as discussed previously.

Another common SIM-related issue is the inability to send SMS messages. This can occur due to improper configuration, such as the module not being set to text mode, invalid phone number formatting, insufficient balance on the SIM card, or power fluctuations. To fix this, configure the module to text mode using AT+CMGF=1, ensure phone numbers are correctly formatted, and verify the SIM card's balance. Adding bypass capacitors near the SIM module can stabilize voltage during high current draws. A 470uF capacitor should work fine. In normal cases this isn't needed.

The button, a critical input component, can also exhibit problems. If false presses are detected, this is often due to electrical or mechanical noise. Adding a 0.1μF capacitor across the button terminals or implementing software debouncing in the Arduino code can mitigate this issue (a debouncing capacitor is included in the PCB design) Inspect and re-solder connections and configure the button pin as INPUT_PULLUP in the code to ensure a stable signal.

Power supply issues are the most common issues and can manifest as the Arduino failing to boot or the SIM module overheating. Incorrect battery connections (reverse polarity), insufficient current supply, low battery voltage (18650 below 3.5 volts) or voltage mismatches can cause these problems. Ensure proper battery polarity and securely connect JST connectors. For the SIM module, confirm the boost converter outputs a stable 5V and can supply at least 2A. Replace faulty components if overheating persists. Check solder joints

to make sure components are secure and properly soldered to the pcb pads and traces. Inspect the PCB for trace issues if made in house.

For debugging, uncomment the Serial.print lines in the code to monitor real-time operation through the serial monitor. Use a dedicated test sketch (included in the repository) to verify SIM800L functionality and a multimeter to measure current draw during operation and sleep mode. Periodically inspect the PCB for clean and secure solder joints, ensure proper orientation of polarized components, and confirm the polarity of all connectors based on the PCB layout and wire color scheme. Additionally, position the antenna to minimize signal interference and maximize network connectivity. The sim800L should draw about 100-200mA when searching for a network then the current should drop to around 40mA once connected. Reference [1] has more information about troubleshooting the SIM800L.

When debugging using the serial monitor here are some of the important network connection AT responses and errors encountered. When working with the +CREG command on the SIM800L module, you may encounter various registration status codes, each indicating different network registration states. Here's a troubleshooting guide for each:

+CREG: 0,0 (Not registered, no network search): This indicates that the module is not registered to any network and is not attempting to find one. Common causes include a faulty or missing SIM card, a SIM card with a PIN lock (which needs to be disabled), or an insufficient signal in your area. It can also happen if the SIM card is not activated or is incompatible with the module.

+CREG: 0,1 (Registered, home network): This means the module is successfully registered on the home network and can send/receive data. If you're not seeing this status, check if the SIM is inserted correctly, and ensure the module has a stable power supply and good signal strength.

+CREG: 0,2 (Registered, roaming): The module is connected to a network while roaming. This could be normal if you're in an area where your home network's coverage doesn't reach, and you're connected to a partner network. Ensure that roaming is enabled on your SIM card, as some carriers disable roaming by default, and you may need to contact your provider to enable it.

+CREG: 0,3 (Registration denied): This indicates that the network has rejected the module's registration attempt. Causes could include a network incompatibility (e.g., if you're using a 2G-only SIM800L in an area where 2G networks are no longer available), network congestion, or issues with the SIM card (e.g., a suspended account). Ensure that your SIM card is active and supports the necessary network bands.

+CREG: 0,4 (Unknown registration status): This means that the module couldn't determine the registration status, possibly due to network issues, signal problems, or configuration errors. Common causes include poor or no signal, incorrect network settings, or a malfunctioning SIM card. Try moving to a different location with better signal, or reset the module with AT+CFUN=1,1 to force a restart and reattempt registration.

By following these troubleshooting steps, users can address most hardware issues encountered with the IoT emergency button device, ensuring long-term reliability and functionality. Regular maintenance, such as inspecting battery connections and testing the device periodically, can further enhance performance.

3.3 Regulatory Considerations for the Device

The deployment of this GSM-enabled device involves critical regulatory requirements to ensure legal compliance and uninterrupted functionality. Given the nature of GSM modules like the SIM800L, users must adhere to specific telecommunication laws and standards to avoid operational issues. This section provides a detailed overview of these regulations, their importance, and guidelines for registering the device with the local authorities.

3.3.1 Telecommunication Module Registration:

GSM modules, unlike standard consumer phones, require additional steps to be authorized for use on local networks. Consumer phones are pre-registered with unique serial numbers (IMEI) recognized by the government, allowing them to connect seamlessly to the network. GSM modules like the SIM800L, however, are not always pre-registered and require manual authorization to avoid being flagged as unauthorized devices.

When importing these modules from abroad, the Ministry of Telecommunications (MoT) of the user's country must be notified. Failure to disclose and register these modules can result in the device being temporarily allowed to connect, maybe once only, flagged as

unauthorized, and subsequently banned from the network. This scenario caused significant delays in this project, as unregistered SIM800L modules were flagged and disconnected, rendering the device non-functional.

3.3.2 Vendor Responsibility and Local Purchases

For users purchasing these modules locally, it is essential to confirm whether the vendor has already registered the modules with the MoT. Some vendors adhere to the necessary regulations and pre-register the modules during importation, but others do not. Buyers should verify this information before purchasing. If the modules are not pre-registered, they may face connectivity issues, including temporary network access followed by a ban due to non-compliance. Check the functioning of the SIM module at the vendor's shop by simply connecting it to power. There is no need to connect TX or RX to check it is connecting to the network as long as it has a SIM card in it. The light blinking indication timing proves its connectivity as seen in reference [x].

3.3.3 Registration Process for GSM Modules (if needed)

1. Obtain Proof of Purchase: Ensure that you have a valid receipt or invoice from the vendor detailing the module purchase.

2. Gather Required Documentation: Obtain the datasheet of the GSM module (such as the SIM800L), which contains technical specifications and can often be downloaded from the manufacturer's website or referenced in this document [2]. Even this documentation might be useful.

3. Submit Registration Application: File an application with the local Ministry of Telecommunications (MoT), including the receipt, module datasheet, and a description of the intended use of the device.

4. Await Authorization: The MoT will review the application and, upon approval, authorize the module to connect to the network.

This process ensures that the device complies with local telecommunication laws, preventing delays and disruptions in its operation.

3.3.4 Impact of 2G Network Deprecation

The SIM800L module operates exclusively on 2G networks, which are being phased out in many countries in favor of more advanced 4G and 5G technologies. This deprecation poses a risk to the long-term functionality of the device, as it relies entirely on 2G connectivity to send SMS messages.

Users must confirm whether their local network still supports 2G services and remain vigilant about future updates from telecommunication providers regarding the discontinuation of 2G services. In the event of a complete 2G phase-out, the device would require a hardware upgrade to use newer GSM modules compatible with 4G or 5G networks.

3.3.5 Commercial Use Regulations

For commercial deployment, it is critical to understand and adhere to the local telecommunication and electronic device laws. Commercial use often imposes stricter requirements, including additional certifications or inspections.

Users must verify:

- Whether the GSM module complies with telecommunication standards.

- If the device design and operation align with electronic safety regulations.

- Any additional licenses required for commercial use.

Safety and Security Considerations

Regulatory oversight of GSM modules exists for security reasons. Such modules can potentially be misused for malicious purposes, such as remote detonation devices or unauthorized network access. Ensuring proper registration and adherence to local laws mitigates these risks and ensures the device operates within the bounds of the law.

The use of GSM-enabled devices like this one comes with specific regulatory responsibilities. Users are advised to proactively address registration and compliance requirements before deploying the device. Whether purchasing modules locally or importing them, proper documentation and adherence to telecommunication laws are critical. Additionally, keeping abreast of changes in network standards, particularly regarding 2G phase-outs, is essential to ensure the device remains operational in the future. For users

intending to deploy this device commercially, further legal and regulatory checks are advised to avoid potential liabilities.

3.4   Future Improvements and Contributions

This project represents an essential step toward developing a reliable IoT emergency meeting scheduling device. While the current implementation achieves its primary objectives, there are numerous opportunities for improvement in both the hardware and software components. These enhancements will not only refine the product's functionality but also make it more adaptable to future needs. By making this project open source, the intention is to encourage collaboration and innovation from the wider community, fostering continuous development and improvement.

One of the primary areas for enhancement lies in the backend. Introducing robust error-handling mechanisms would allow the system to recover gracefully from issues such as API failures, network disruptions, or invalid data submissions. Additionally, a standalone website can be developed to oversee backend operations, providing a centralized interface for users to monitor events, configure devices, and manage schedules. To complement this, a frontend dashboard could present real-time updates on button presses, scheduled meetings, and system health metrics, significantly improving user experience and accessibility.

From a hardware perspective, future designs could focus on creating a more compact and ergonomic enclosure. This would involve optimizing the layout of internal components such as batteries, the PCB, and the button. Enhancing the PCB design itself is another critical improvement area. Being one of the project's first PCB iterations, there is room for better trace layouts, reduced electrical noise, and more efficient use of space. Additionally, incorporating reverse polarity protection for the batteries would prevent potential damage from incorrect installation. Including features such as overcharge and deep-discharge protection would further improve the device's reliability and safety.

Adapting to evolving network technologies is another significant consideration. The current design uses the SIM800L, which operates on 2G networks. As 2G services are phased out in favor of 4G and 5G networks, integrating modules compatible with newer standards will ensure the device remains functional in the long term. Moreover, the battery monitoring circuit, which utilizes the A0 pin, a 100k resistor, and a custom code, is already

included in the repository. Future iterations could improve upon this by incorporating more precise monitoring ICs to deliver detailed battery analytics.

As an open-source initiative, this project invites developers, engineers, and enthusiasts to contribute their expertise to overcome existing limitations and introduce new features. The battery monitoring section and related functionalities, while not elaborated upon in this report, are available in the repository for further exploration and refinement. By encouraging open collaboration, the project aims to harness the creativity and skills of the community, transforming it into a robust, scalable, and highly adaptable product for emergency meeting scheduling and beyond.

4. CONCLUSION

In conclusion, this open-source IoT device represents a significant advancement in emergency communication solutions, combining simplicity, scalability, and security to meet the diverse needs of industries ranging from healthcare to corporate environments. With its user-friendly design, long-lasting performance, and robust security features, the device promises to revolutionize how urgent meetings are scheduled and coordinated. The open-source approach not only makes the device accessible to a broader audience but also encourages collaboration and innovation, allowing for continuous improvement and customization. By offering a cost-effective, reliable, and flexible solution, this project has the potential to transform emergency communication practices and enhance operational efficiency across various sectors.

REFERENCES

[1] SIM800L with Arduino, https://lastminuteengineers.com/sim800l-gsm-module-arduino-tutorial/

[2]SIM800L Datasheet,

https://www.makerhero.com/img/files/download/Datasheet_SIM800L.pdf

[3]Relay Datasheet, chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://img.ozdisan.com/ETicaret_Dosya/445
413_4369639.pdf

[4] MT3608 Datasheet, chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.olimex.com/Products/Breadboar
ding/BB-PWR-3608/resources/MT3608.pdf

[5] Arduino Sleep Mode, https://www.instructables.com/A-Guide-to-Putting-Your-Arduino-to-Sleep/

[6] Google Calendar API, https://developers.google.com/calendar/api/guides/overview

[7] Twilio API, https://www.twilio.com/docs/serverless/functions-assets/quickstart/receive-sms

APPENDIX A

Arduino working code:

```
#include <SoftwareSerial.h>
#include "LowPower.h"

// Pin Definitions
const int buttonPin = 2;     // Button connected to pin 2
const int mosfetPin = 6;     // MOSFET control pin connected to pin 6
const int ledPin = 13;        // LED for indication
bool buttonPressed = false;  // Tracks if button is pressed
```

```
String ClientID = "001"      // for scalability change the client button if
there are many buttons.
// Create software serial object to communicate with SIM800L
SoftwareSerial mySerial(3, 4); // SIM800L Tx & Rx connected to Arduino pins #3
& #4

// Interrupt Service Routine (ISR) for waking up the Arduino
void wakeUp() {
  buttonPressed = true; // Set the flag to indicate button press
}

void setup() {
  // Configure pins
  pinMode(buttonPin, INPUT_PULLUP); // Button with pull-up resistor
  pinMode(mosfetPin, OUTPUT);       // MOSFET control pin
  pinMode(ledPin, OUTPUT);          // LED pin for feedback
  digitalWrite(mosfetPin, LOW);     // Ensure MOSFET starts off
  digitalWrite(ledPin, LOW);        // Ensure LED starts off

  // Begin serial communication
  //Serial.begin(9600);
  mySerial.begin(9600);

  //Serial.println("System Initialized. Waiting for button press...");
}

void loop() {
  // Attach interrupt to the button pin for wake-up
  attachInterrupt(digitalPinToInterrupt(buttonPin), wakeUp, FALLING);

  // Enter power-down sleep mode until button is pressed
  LowPower.powerDown(SLEEP_FOREVER, ADC_OFF, BOD_OFF);

  // Wakes up here when the button is pressed
  detachInterrupt(digitalPinToInterrupt(buttonPin)); // Disable interrupt
  if (buttonPressed) {
    //Serial.println("Button Pressed! Waking up...");
    buttonPressed = false;

    // Indicate activity with LED
    digitalWrite(ledPin, HIGH);
    delay(500);
    digitalWrite(ledPin, LOW);

    // Power on SIM module
    digitalWrite(mosfetPin, HIGH);
    //Serial.println("Powering on SIM module...");
    delay(2000); // Allow SIM module to stabilize
```

```
    // Check network connection and send SMS
    if (waitForNetwork()) {
      sendSMS();
      delay(5000); // Wait a few seconds after sending the SMS
    }

    // Turn off SIM module
    digitalWrite(mosfetPin, LOW);
    //Serial.println("SIM module powered off.");

    // Indicate return to sleep mode
    digitalWrite(ledPin, HIGH);
    delay(500);
    digitalWrite(ledPin, LOW);
  }
}

// Function to wait for the SIM module to connect to the network
bool waitForNetwork() {
  Serial.println("Checking network connection...");
  for (int i = 0; i < 30; i++) { // Retry up to 30 seconds
    mySerial.println("AT+CREG?");
    delay(1000);
    while (mySerial.available()) {
      String response = mySerial.readString();
      //Serial.println("Network Check Response: " + response);

      // Check if the module is registered to the network
      if (response.indexOf("+CREG: 0,1") > -1 || response.indexOf("+CREG:
0,5") > -1) {
        //Serial.println("Network Connected!");
        return true;
      }
    }
    //Serial.println("Waiting for network...");
  }
  //Serial.println("Network connection failed.");
  return false;
}

// Function to send an SMS
void sendSMS() {
  //Serial.println("Sending SMS...");
  mySerial.println("AT+CMGF=1"); // Set module to SMS text mode
  updateSerial();
```

```
  mySerial.println("AT+CMGS=\"+000000000\""); // Replace with your phone
number
  updateSerial();

  mySerial.print("Message from Arduino: Button was pressed!"); //  //replace
message with "001,BUTTON_PRESS,100%" when using with server to send emails.
Change Client ID if there are more than 1 buttons
  // event type is either a button press or a check battery. the check battery
is for future improvements not included in this code, where the arduino can
monitor the battery voltage. another code is provided for monitoring battery
health.
  mySerial.write(26); // Ctrl+Z to send the SMS
  //Serial.println("SMS Sent!");
}

// Function to forward serial data for debugging
void updateSerial() {
  delay(500);
  while (mySerial.available()) {
    Serial.write(mySerial.read());
  }
}
```