SMS Based Environment Monitoring System (SEMS)

By

Ahmed Yar Khan

Ali Shahzeb Khalid

Ali Aman

Supervised by:

Lt Col Usman Mahmood Malik

Submitted to the faculty of Department of Computer Software Engineering,
Military College of Signals, National University of Sciences and Technology, Islamabad,
in partial fulfillment for the requirements of B.E Degree in Software Engineering.

June 2024

In the name of ALLAH, the Most benevolent, the Most Courteous

CERTIFICATE OF CORRECTNESS AND APPROVAL

This is to officially state that the thesis work contained in this report

"SMS Based Environment Monitoring System"

(SEMS)

is carried out by

Ahmed Yar Khan

Ali Shahzeb Khalid

Ali Aman

under my supervision and that in my judgement, it is fully ample, in scope and excellence, for the degree of Bachelor of Software Engineering in Military College of Signals, National University of Sciences and Technology (NUST), Islamabad.

Approved by

Supervisor

Lt Col Usman Mahmood Malik

Date: _____

DECLARATION OF ORIGINALITY

We hereby declare that no portion of work presented in this thesis has been submitted in support of another award or qualification in either this institute or anywhere else.

ACKNOWLEDGEMENTS

Allah Subhan'Wa'Tala is the sole guidance in all domains.

Our parents, colleagues and most of all, supervisor, **Lt Col Usman Malik** provided invaluable guidance.

We also thank the group members who, through all adversities, worked steadfastly.

Plagiarism Certificate (Turnitin Report)

This thesis has 14% similarity index. Turnitin report endorsed by Supervisor is attached.

Ahmed Yar Khan

358976

Ali Shahzeb Kahlid

358981

Ali Aman

358996

Signature of Supervisor

ABSTRACT

The SMS-Based Environment Monitoring System (SEMS) is a step up from the previous systems for environmental monitoring as it come with modern sensors and wireless connectivity. In contrast to the traditional systems, the SEMS provides a complex and at the same time easy-to-implement solution for the effective control of the critical environmental conditions in different facilities, such as airports and data centers.

The SEMS uses Arduino controller and specific sensor nodes to measure temperature, humidity, air quality, and power consumption effectively and accurately. Each sensor node is equipped with specific sensors: the DHT11 for temperature and humidity, the MQ135 for air quality and sensors for current and voltage. These nodes connect wirelessly via ESP32 modules to a master node hence facilitating data transfer and acquisition.

The master node has a GSM module to send SMS alerts in real-time depending on the data collected to facilitate the response to environmental changes. Moreover, the system has a straightforward graphical user interface that further improves the system's availability and usage by operators. In this regard, this project seeks to enhance environmental monitoring in the CI environments through the development and implementation of this unique monitoring system.

Table of Contents

Table of Figures

Chapter 1: Introduction

The Safety of the different environments has continued to be a cause of concern in the globalized world for governments, regulatory agencies, and the public. As the environmental factors continue to remain a thorn in the horn of the societies, and with the increasing need for monitoring systems, the need for an efficient environmental monitoring measures cannot be over emphasized. Environmental surveillance is a crucial factor that is fundamental in the protection of the lives of people and the structures.

The conventional monitoring techniques involve physical observation and data acquisition by personnel, which is prone to errors and variability. Also, the monitoring tasks have become more complex and large scale, and the personnel involved have concerns about the reliability and efficiency of the current monitoring processes.

To address these challenges, the enhancement of the existing or development of new sensing technologies for environmental applications has become a viable solution. The SMS-Based Environment Monitoring System (SEMS) is one such innovation which encompasses a complete hardware and software solution to monitor different parameters of the environment accurately and effectively. The SEMS system therefore seeks to improve monitoring of the environment as well as the overall environmental status through the use of sensor technology.

This thesis aims at identifying the growth, integration, and assessment of SEMS system as an innovative tool in environmental management.

1.1 Overview

In the contemporary society that is characterized by high levels of digitization, numerous settings play a significant role in providing the interface between technology and safety, especially

as the levels of monitoring and surveillance rise. Due to this, the conventional monitoring approaches are limited in terms of their efficiency and effectiveness. Sensor technology is a rapidly developing area that can help to improve the current state of environmental monitoring and contribute to the development of new methods for data acquisition and analysis.

The SMS-Based Environment Monitoring System (SEMS) defines a new frontier in environment monitoring as a breakthrough invention. With the help of the sensor modules for the temperature, humidity, air quality, and electrical parameters, SEMS provides the objective and uniform method of the environment controlling independently. In comparison with the manual methods, the use of SEMS helps to minimize the chance of errors, thus increasing the general reliability of the monitoring processes.

It is important to note that there is growing awareness of environmental issues hence the need to adopt proper monitoring procedures. Current systems are not always capable of adapting to new changes in the environment hence there are shortcomings in monitoring. To these challenges, SEMS offers solutions since it offers timely data analysis, which helps stakeholders to act on changes and prevent risks.

Besides, the implementation of SEMS in diverse settings provides added advantages apart from data acquisition. Through the optimization of monitoring processes, SEMS decreases response times and consequently increases the operational efficiency. Also, SEMS has the advantage of cutting costs by reducing the amount of data that has to be collected and analyzed manually.

The adoption of SEMS can be considered as a positive shift in strengthening the measures towards monitoring the environment. With the help of sensors and advanced design, SEMS provides a solution to contemporary monitoring issues. It is the purpose of this thesis to analyze the creation,

application, and outcomes of SEMS, and how it can revolutionize the field of environmental monitoring and improve safety measures in different areas.

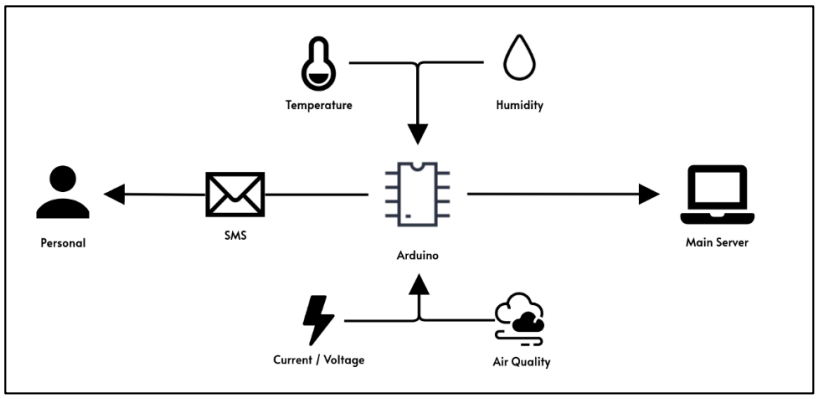

Figure 1 - General Overview of SEMS

1.2 Problem Statement

In current environmental monitoring systems, the use of conventional approaches to data collection and analysis for monitoring operations proves to be problematic and ineffective in many ways, consequently leading to doubts over the efficiency and feasibility of the monitoring initiatives. The following highlights illustrate the shortcomings of existing environmental monitoring systems:

1. Due to the reliance on manual data collection, the demand for manpower is increased hence experiencing high chances of running out of staff and thus inefficiency in operations.

2. The absence of automation in the monitoring process translates into delayed real time data feedback since manual collection cannot efficiently cope with the changing environmental conditions.

3. Lack of regularity in the monitoring intervals and covered areas in the different monitoring points has also led to some of the gaps in monitoring and may fail to detect some of the changes in the environment due to the inconsistencies in the monitoring efforts.

4. Lack of application of sensor technologies in monitoring infrastructure requires constant maintenance and calibration exercises that make it expensive and require more resources.

5. These manual methods of monitoring may result in high chances of human intervention and therefore increases the probability of making wrong decisions which may affect the overall results of the environmental monitoring efforts.

1.3 Proposed Solution

The idea of our proposed solution can be summarized into the following major goal; to facilitate the automation of the current process of monitoring the environment through the integration of a SEMS using sensor technologies. The proposed solution is to overcome the shortcomings of the existing manual approach and achieve higher levels of environmental monitoring, providing timely and accurate information in all the monitoring points.

1.4 Working Principle

The project mainly covers the working model of sensor data acquisition along with the communication protocols. The work is divided into a number of sections, where each section is connected to the succeeding one.

- Sensor Data Collection
- Data Processing
- Decision making based on Processed Data
- Integration
- GUI Presentation

1.4.1 Sensor Data Collection

The essence of the project is the data acquisition from the different sensors placed in the various monitoring sections. Sensors include DHT11 for temperature and humidity, air quality sensor MQ135, and ZMCT103C for current and voltage.

1.4.2 Data Processing

The gathered data from the sensors is analyzed to extract necessary information and find correlations. This particular step of processing is significant to be able to draw an informed result from the values of the environmental parameter under consideration.

1.4.3 Decision Making

These include the decisions available in regards to the environment, and the subsequent actions to be taken on the processed data. This includes the sending of alarms or notification regardless of the readings if it falls out of the normal range.

1.4.4 Integration

Moreover, all the analyzed modules are integrated into one system that provides direct communication and cooperation of the master and slave nodes. This integration as an aspect ensures that it facilitates the sharing of data and the process of decision making.

1.4.5 GUI Presentation

The obtained data from the sensor nodes is presented in the end user through the formation of an interface to the external environment in the form of a graphical user interface (GUI). The application allows users to observe the current environmental parameters online, select the necessary data from the sensors, and configure the parameters of monitoring if required.

1.5 Objectives

1.5.1 General Objectives

The main goal of SEMS is to provide the efficient environmental monitoring system by using the sensor nodes and wireless communication technology. This system aims to:

- Design a viable method for monitoring the environment based on sensor information to identify conditions in real-time.
- Integrate a network of sensor nodes that allow for capturing and transmitting data concerning temperature, humidity, air quality and power status.
- Design an intuitive and easy to use graphical front end to capture and display the data collected by the sensor nodes for the environment.
- Guarantee the expansion of the monitoring system to reflect the different environmental monitoring requirements.
- Ensure that the system provides automated alerts/notifications to respond quickly to environmental changes/events.

1.5.2 Academic Objectives

- Place and install the sensor nodes with DHT11, MQ135 and ZMCT103C sensors for the purpose of environmental monitoring.
- Design methods for handling and analyzing data gathered through sensors to understand the state of the environment.
- Incorporate the sensor nodes with the rest of the network to allow for proper transfer of data within the framework.

- Investigate methods used in the display and representation of environmental information in a way that can be easily understood from a GUI perspective.
- Identify ways of reducing the power consumption and the overall usage of the resources within the monitoring system to enhance its operational time.

1.6 Scope

The scope of this project also encompasses the monitoring and surveillance within the data centers through SEMS. It is designed as a flexible software that can solve the problem of detecting various parameters of the surrounding environment by analyzing data from sensors. The system's goal is to effectively observe the environment of data centers to support safety and security of important infrastructures. The applications of the SEMS within data centers include:

- **Early Anomaly Detection**: Alerting the concerned authority on time for detecting any changes in the environment like the temperature, humidity, and quality of air.
- **Enhanced Safety Measures**: Supervising conditions that affect the environment in which data center equipment and workers operate.
- **Operational Readiness**: Helping in the planning of and managing contingencies that may occur within the environment and affect data center.
- **Preventive Measures**: Helping in reducing the occurrence of environmental threats, identifying risks that may cause harm to the environment and then acting on them to minimize their impact.
- **Safeguarding Data Integrity**: Increasing the level of protection for the general information by controlling environmental conditions to avoid breakdowns and failures of equipment.

1.7 Deliverables

The following is the breakdown of the product of the SMS-Based Environment Monitoring System Project:

- Software Requirements Specification (SRS)
- Software Design Document (SDD)
- Thesis
- Executable Project File
- Hardware Components
- Installation Guide
- User Manual
- Technical Requirements Document

1.7.1 Comprehensive Security Overview

This system offers the versatile method to monitor the all-round of an environment, in which various send SMS-based environment monitoring technique is used for analyzing the information about the various sensors that are installed within the environment of the data center.

1.7.2 Environmental Anomaly Detection

The system is programmed to study the shifts of parameters such as Temperature, humidity, Air quality, and electricity consumption with advanced algorithms. It can also

identify deviations from normal processes, and this enables one to counter check for problems that are likely to happen in the future.

1.7.3 Hazard Identification

It also can identify some hazards in data center environment like temperate increase, low quality of air, and changes in voltage or current. This way it assists in managing the risks and in introducing measures that would have an impact on the security of data center structures.

1.7.4 Real-time Alerts

The system is able to be on the alert the moment there is a change in the environment in the data center and through the use of the SMS notification, the people in charge of the data centers are notified. These alerts enable related actions and countermeasures to be taken where there is danger of equipment harm or system out of reach.

1.7.5 Historical Data Analysis

The system also records and stores historical data of the environment which can be used for trend analysis and for scheduling for maintenance. In this way, previous trends are taken into consideration, and the utilization of resources, as well as the effectiveness of data center management, is enhanced.

1.8 Relevant Sustainable Development Goals

The project is aligned with the following sustainable development goals – **SDG 9, SDG 11 and SDG 13.**

1.8.1 SDG 9: Industry, Innovation, and Infrastructure

- The project helps to achieve SDG 9 as it focuses on innovation in the environmental monitoring systems and data centers for the improvement of industry practices and infrastructure.
- The system incorporates sophisticated sensor technologies and wireless communication modules that enhance the creation of new monitoring systems, thus, promoting technology advancement in the industry.
- It enhances infrastructure durability by offering data on the status of the external environment of data centers to allow for timely management of a range of aspects of infrastructure resources.

1.8.2 SDG 11: Sustainable Cities and Communities

- The project contributes to the SDG 11 because sustainable practices in data center management are crucial to the construction of resilient societies.
- By conducting environmental monitoring and management, the system reduces the harm of data center activities to the environment and supports the development of sustainable urban environment.
- Thus, by providing the stability and reliability of data center operations, the project increases the protection of communities and preserves digital facilities and services.

1.8.3 SDG 13: Climate Action

- The project contributes to the achievement of Sustainable Development Goal 13 by effectively managing climate change through proper environmental monitoring in data centers.

- It is used to monitor temperature, humidity and air quality, which in turn helps the operators of data centers to control energy consumption and minimize carbon footprint.
- It assists climate action by providing data centers with best practices to reduce climate impact including optimized cooling and renewable energy integration.

1.9 Structure of Thesis

Chapter 2: Literature Review - This section presents the current knowledge and context of the research that informs the project.

Chapter 3: Software Requirement Specification - Documents the requirements to be implemented in the development of the SMS-based environment monitoring system in detail.

Chapter 4: System Design & Architecture - Explains the software architecture of the SMS-based environment monitoring system and provides diagrams and a description of the database structure.

Chapter 5: Hardware Design - Includes the description of the circuit design of the monitoring system, the integration of sensors and the communication between the hardware parts of the system.

Chapter 6: Software Design - Outlines the various aspects of the software design of the monitoring system with regard to the server interfaces and communications protocols.

Chapter 7: Conclusion and Future Work - Presents the overall results and the conclusions reached at the end of the project, also based on the research and development work conducted.

Chapter 8: Future Work - Proposes possible advancements to the SMS-based environment monitoring system and suggests further research areas.

Chapter 2: Literature Review

The analysis of the application of sensor-based monitoring systems in data centers demonstrates a progressive improvement in infrastructure management. This literature review focuses on the industrial environment, the common practices, and the relevant literature to provide a comprehensive understanding of the environment and challenges involved in implementing an SEMS. The analysis of the investigation is based on the following components:

- Industrial Background
- Existing solutions and their drawbacks
- Research Papers

2.1 Industrial background

Environmental monitoring is critical when it comes to data centers because it helps to establish the overall integrity of the operation while protecting the hardware. Modern industries rely heavily on data centers to perform diverse operations, which calls for appropriate measures to enhance efficiency and security. However, conventional monitoring paradigms have limitations in dealing with the changing dynamics of data center settings.

In this context, the data center industry is experiencing increasing interest in new monitoring technologies, especially those based on sensors. Leading environment monitoring system providers are now at the forefront of creating better solutions to support the strengthening of data center operations. In an effort to encourage the development of more environmentally friendly and energy efficient data centers, the incorporation of sensor-based monitoring systems, including SEMS, marks a major improvement in data center efficiency and reliability.

2.2 Existing Solutions and their Drawbacks

In the context of data center monitoring, traditional approaches involve routine inspections and simple sensor systems to assess environmental parameters. The first one is a precautionary method where technicians have to check the system periodically and use normal sensors to measure such factors as temperature and humidity.

- **Process:** Hands-on assessments involve physical examination of the data center environment by the technicians, using simple instruments such as thermometers and hygrometers to gauge environmental conditions.
- **Advantages:** The practical tests enable the monitoring and appraisal of environmental conditions as it is done by the technicians in real time.
- **Limitations:** Like most practical examinations, hands-on assessments are invasive and require much time, which may lead to more time before potential problems are detected. Basic sensor technology may be less sensitive and may be unable to capture small changes in the environmental conditions. However, the practical examinations are prone to human factors and might fail to notice some minor deviations.

The primary monitoring solutions that have been in use include manual assessments and basic sensor technologies that came with some drawbacks and thereby called for better solutions. New technologies, including sensors and IoT solutions, seek to increase the effectiveness of the monitoring process.

These systems employ multiple sensors and real-time data analysis, which help to minimize risks and prevent failures, thus improving data center efficiency and availability. This transition to

smarter monitoring solutions is indicative of the trend towards automation and the digitization of data center operational environments that has taken hold in the industry.

2.3 Research

The exploration for SEMS encompasses the following stages:

1. **Investigation of Sensor Technologies**: This will involve testing different sensors such as the DHT11 for temperature and humidity, MQ135 for air quality, and ZMCT103C for power monitoring and comparing their efficiency in measuring different environment aspects.

2. **Comparison of Single-Board Computers (SBCs)**: Compare various SBCs like Arduino and Raspberry Pi for its suitability to host the monitoring system by evaluating its processing capability, connectivity and power consumption.

3. **Analysis of Related Research Papers**: To get aware of the new trends, methods and finding related to the SMS-Based Environment Monitoring Systems an analysis of various research papers in the same domain of our FYP was conducted. This entailed conducting a literature review of the existing literature from peer-reviewed journals, conference proceedings and other publications in order to design and deploy the monitoring system.

4. **Studies on Sensor Data Processing**: Research techniques of dealing with sensor data such as data acquisition, analysis, and visualization that can be useful in environmental monitoring.

5. **Applications in Data Center Monitoring**: Discuss the use of sensor technologies in data centers and how temperature, humidity, and power usage are some of the sensors that need to be monitored to ensure that the equipment is running efficiently.

6. **Integration of Sensor Data with Network Monitoring**: Review studies done on the use of sensors and the integration of the sensor data into the network monitoring systems to get an overall picture of the data center and get early warning signs of problems.

7. **Evaluation of Disaster Preparedness:** Examples include research on disaster preparedness in data centers, including the monitoring of sensor data in the event of power outages or equipment failure to reduce the risks associated with loss of data integrity.

2.3.1 Investigation of Sensor Technologies

Different sensors are essential components of SEMS as they enable the real-time measurement of environmental parameters like temperature, humidity, air quality, and power use. Of all the sensors used in this project, the DHT11 is used for sensing temperature and humidity, whereas the MQ135 is used for sensing the quality of air in the environment. Moreover, the ZMCT103C sensor is chosen to measure the power for the voltages and currents as well. Such choices are made based on attributes such as accuracy, reliability, and the ability to interface with the said monitoring system. Much work is done in understanding the properties of all the used sensors, calibration procedures and how data acquisition is done with the aim of getting the best performances under different weather conditions. Thus, the location of sensors, the frequency at which data from these sensors has to be collected, and the procedures that are followed in transmitting the data are well evaluated to facilitate the inclusion of the sensors into the overall structure of the monitoring system.

2.3.1.1 Comparison of Temperature Sensors

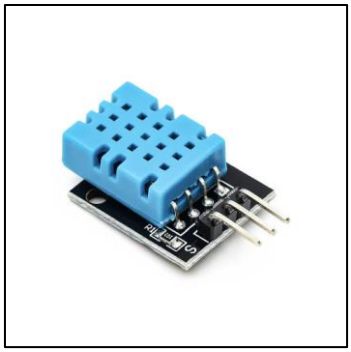

Figure 2 – DHT11

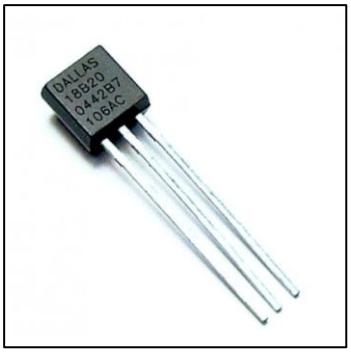

Figure 3 – DS18B20

Figure 4 - LM35

1. **DHT11 vs. DS18B20:**

Feature	**DHT11**	**DS18B20**
Range	20-90%RH, 0-50 ℃	-55°C to +125°C
Accuracy	±5%RH, ±2℃	±0.5°C (at 25°C)
Response Time	6-15s	750ms-10s
Interface	Analog/Digital	Digital
Calibration	Factory Calibrated	Calibrated on Demand

2. **DHT11 vs. LM35:**

Feature	DHT11	LM35
Range	20-90%RH, 0-50 ℃	−55゜to +150℃
Accuracy	±5%RH, ±2℃	±0.5℃ (at 25℃)
Response Time	6-15s	-
Interface	Analog/Digital	Analog
Calibration	Factory Calibrated	Calibrated on Demand

Between the LM35 and DS18B20 temperature sensors, the DHT11 sensor has an advantage of having a dual interface. The DHT11 sensor is pre-calibrated to ensure that users get accurate temperature and humidity results as soon as they start using the sensor. Such features make the DHT11 sensor a versatile and reliable option for measuring temperature and humidity.

2.3.1.2 Comparison of Air Quality Sensors

Figure 5 – MQ135

Figure 6 – CCS811

Figure 7 - BME680

1. **MQ135 vs. CCS811:**

Feature	**MQ135**	**CCS811**
Gas Sensing Capability	CO2, NH3, VOCs	CO2, TVOC
Burn-in Time	48 hours	48 hours
Measurement Range	10~1000ppm	400~8000ppm
Response Time	10s-5m	30s-2m
Interface	Analog/Digital	Digital

2. **MQ135 vs BME680:**

Feature	**MQ135**	**BME680**
Gas Sensing Capability	CO2, NH3, VOCs	CO2, TVOC
Burn-in Time	48 hours	48 hours
Measurement Range	10~1000ppm	Provides Res Value
Response Time	10s-5m	1-8s
Interface	Analog/Digital	Digital

MQ135 is more preferable than CCS811 and BME680 sensors because of its expanded range of gas detection, including CO2, NH3, and VOCs. The sensor also has a faster response time than CCS811 with the response time of 10sec to 5 min. These features make the MQ135 sensor more suitable and effective in detecting different gases at different response times.

2.3.1.3 Comparison of Voltage/Current Sensors

Figure 8 - ZMCT103C

Figure 9 - ACS712

Figure 10 - INA219

1. **ZMCT103 vs. ACS712:**

Feature	**ZMCT103C**	**ACS712**
Measurement Range	0A to 10A	-30A to +30A.
Output	Analog	Analog
Accuracy	±1%	±1.5%
Response Time	-	5µs
Installation	Invasive	Invasive

2. **ZMCT103C vs INA219:**

Feature	ZMCT103C	INA219
Measurement Range	0A to 10A	±3.2A (Prog Gain)
Output	Analog	Digital
Accuracy	±1%	±0.5%
Response Time	-	-
Installation	Invasive	Non-Invasive

The ZMCT103C sensor is more preferable than ACS712 and INA219 sensors because of the wider dynamic range and high accuracy. The current output of the ZMCT103C is 0-10A with an accuracy of ±1% making it more suitable for voltage and current measurements. Also, analog output is easier to interface with other systems and thus, makes it the best for voltage/current sensing in SEMS.

2.3.2 Comparison of Single-Board Computers (SBCs)

In the context of SEMS, selection of SBCs plays an important role in determining the performance, communication interfaces, and power consumption of the system. Various SBCs including Arduino and Raspberry Pi, are assessed in terms of their processing power, memory, I/O interfaces, and compatibility with sensor modules. The comparison includes performance tests that aim to compare the computational capabilities and power usage of each SBC when operating in a standard manner. Furthermore, it focuses on the factors such as the simplicity of coding, support from a community, and the compatibility of additional components for extension and modification. The chosen SBC must be able to meet the real-time data processing, and interaction with the sensors as well as the human interface of the monitoring system. The above comparative analysis helps in identifying the most appropriate SBC that can be used as the core processing unit for the SMS-Based

Environment Monitoring System to have reliability, scalability and economical costs of deployment.

1. **Arduino UNO vs Raspberry Pi 4:**

Figure 11 – Arduino UNO

Figure 12 – Raspberry Pi 4

Feature	**Arduino Uno**	**Raspberry Pi 4 Model B**
Processing Capabilities	16 MHz ATmega328P microcontroller	1.5 GHz quad-core ARM Cortex-A72
Memory	2 KB SRAM, 32 KB Flash	4 GB LPDDR4-3200 SDRAM
Input/Output Interfaces	14 digital I/O pins, 6 analog I/O pins, UART, SPI, I2C	40 GPIO pins, UART, SPI, I2C, HDMI, USB
Connectivity Options	USB, UART	Ethernet, Wi-Fi, Bluetooth, USB
Compatibility with Sensors	Compatible with various sensor modules via GPIO pins	Compatible with sensor modules via GPIO pins, USB, I2C
Power Efficiency	Lower power consumption	Moderate power consumption
Ease of Programming	C/C++ programming language	Python, C/C++, Java, and more
Peripheral Expansion	Limited expansion capabilities	Extensive expansion options via GPIO pins, USB
Cost	Affordable	Moderate

The Arduino Uno and Raspberry Pi 4 Model B are both compared as the possible core processing unit of SEMS. Despite the fact that Raspberry Pi is more powerful in terms of CPU, RAM and connectivity the Arduino Uno is preferred for its simplicity, stability and low power consumption. Due to the high compatibility of the Arduino Uno with different sensor modules, and owing to the fact that C/C++ language is easier to program than other languages, Arduino Uno is more suitable for real time data acquisition and interfacing with sensor devices. Moreover, the Arduino community offers a great deal of support and tools for development, so that the implementation and deployment of SEMS should not pose any problems.

2.3.3 Analysis of Related Research Papers

2.3.3.1 GSM-Based Remote Wireless Automatic Monitoring System

1. **Implementation of GSM-Based Remote Monitoring System:**
    - The paper outlines the design and installation of a Remote Monitoring System using GSM technology. It also provides real time information of the field that include temperature, humidity and moisture content of the soil.
    - The use of GSM technology makes it possible to communicate wirelessly over long distances and this makes it possible to transmit data from far off locations to a central monitoring station. This offers advantages of mobility and ease in capturing information in the field without necessarily being in the monitoring site.
2. **Sensor Integration and Data Collection:**
    - To capture information from the field, the monitoring system combines several sensors such as temperature sensors, humidity sensors, and soil

moisture sensors. These sensors constantly measure the conditions of the environment and then relay the information to the main control hub through a wireless connection.

- Data generated from the sensors are analyzed to give information on environmental conditions in order to support decision making in agricultural practices and environmental monitoring.

3. Automation and Remote Access:

- The last characteristic of the GSM-based monitoring system is Automation and Remote accessibility. The system is fully automated to monitor and collect data and send it to the required destination without any human interference.
- By the help of a friendly user interface, users can be able to view the current data and the monitoring parameters from the field with ease and compare the current conditions, monitor trends and also be notified in case there is an abnormality or an event occurs.

2.3.3.2 Wireless Sensor Network for Monitoring & Control

1. Implementation of Wireless Sensor Network (WSN):

- This paper describes the implementation of a WSN for monitoring and controlling of environmental conditions through Arduino microcontrollers. The WSN comprises of multiple numbers of sensor nodes that are deployed in the monitoring area, which are connected wirelessly with the central control unit.

- Arduino microcontrollers are central to each sensor node as they collect data from different environment sensors and allow for wireless transmission between nodes and the master controller.

2. **Sensor Integration and Environmental Monitoring:**
    - The WSN includes temperature sensors, humidity sensors and light sensors for measuring the important environmental conditions. These sensors are always collecting data on the environment to give an instantaneous reading on temperature fluctuations, humidity, and light intensity.
    - The information gathered by the sensor nodes is then sent wirelessly to the base station where the information is then processed and sorted. This makes it possible to conduct proper monitoring of the environment and to be able to provide quick response to any changes or fluctuations in the environment.
3. **Control and Actuation Capabilities:**
    - Besides, the WSN has control and actuation for controlling and acting to the changes of the environmental factors. The central control unit can also transmit signals to the sensor nodes to change the parameters of light intensity or temperature depending on the set values or user requests.
    - This control functionality increases the flexibility of the WSN and increases the ability to adapt to the environment and create automation in various spheres, including agriculture, smart buildings, and environmental monitoring.

Chapter 3: Software Requirements Specification

3.1 Overall Description

3.1.1 Product Perspective

SEMS is a new, standalone product that was developed to work independently but at the same time, it is compatible with the current structure. It is not a replacement of any existing systems but it does bring in a fresh concept of real-time monitoring. The product operates within the following context:

- **Independence:** The monitoring system is independent and does not require significant changes to the hardware and software of the bus or integration into them.
- **Integration with Microcontroller/SBC and GSM Module:** The components of the system are the microcontroller or Single Board Computer and the GSM module. These comprise of communication, data processing and the generation of SMS alerts.
- **Wireless Sensor Communication:** Environmental sensors are connected to the microcontroller/SBC wirelessly which makes the system easily scalable and modular.
- **User Interface:** It comprises a software application that enables a user to monitor and control the devices. This interface is an essential tool when it comes to communication and decision making.

- **Real-time SMS Alerts:** The last but not the least part of the system is real time SMS alerts which are an integral part of the entire system. These notifications are passed to specific officials for prompt action when there is an issue or a failure.

Though the SMS-based monitoring system is independent in a way, its functionality depends on the smooth integration of the main components of the system and the ability to deliver SMS alerts to the concerned personnel. The following is the System Diagram of the SEMS System:

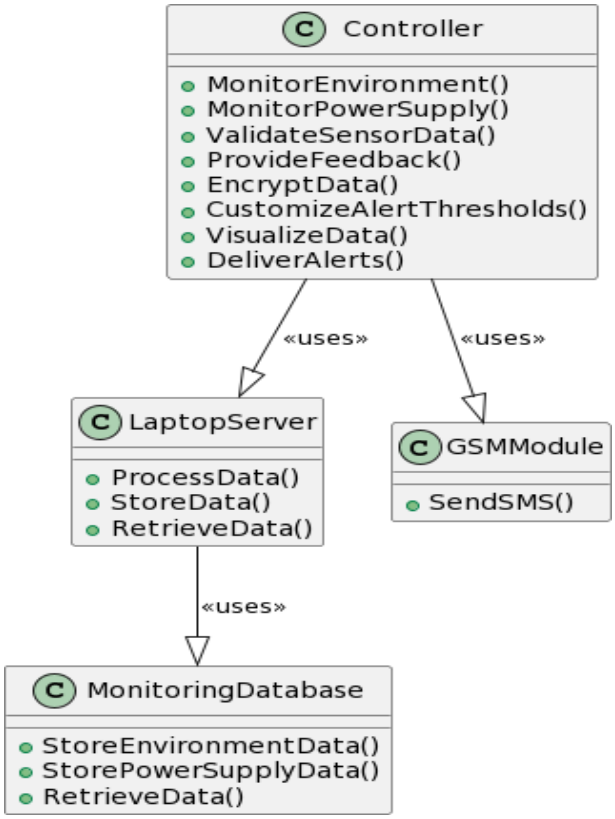

Figure 13 - System Diagram

3.1.2 Product Functions

The following are critical functions of SEMS to provide a holistic real-time monitoring and alerting function:

- **Environmental Monitoring:**
    - Monitoring of temperature, humidity, and air conditions on a real-time basis.
    - Wireless communication between sensors for seamless data transfer.
    - Continuous analysis of environmental parameters to detect anomalies.
- **Power Supply Monitoring:**
    - Monitoring and assessment of AC mains, backup generators, and solar power sources.
    - Real-time tracking of power supply status and identification of potential failures.
    - Integration with the monitoring system for comprehensive power infrastructure analysis.
- **Alert Generation and Communication:**
    - Swift generation of real-time SMS alerts in response to environmental or power supply anomalies.
    - Configurable alert thresholds for different parameters.
    - Seamless integration with a GSM module for efficient and reliable SMS communication.
- **User Interface:**

- Development of a user-friendly software application/web app for monitoring and management.
- Intuitive graphical interface providing a visual representation of environmental and power supply data.
- Customizable dashboard for personalized monitoring preferences.

- **Comprehensive Testing:**
  - Rigorous testing protocols for both hardware and software components.
  - Functional testing to ensure all monitoring and alerting features operate as intended.
  - Reliability testing to assess the system's performance under various environmental conditions.

- **Logging and Reporting:**
  - Logging of environmental and power supply data for historical analysis.
  - Generation of detailed reports summarizing system performance and detected anomalies.
  - Logs that are easily visible and can be used for diagnostics and fine-tuning.

All these functions in total contribute to the main aim of developing an effective and reliable SMS based monitoring system for the environmental and power supply monitoring in various situations.

3.1.3 User Classes and Characteristics

In SEMS, two primary user classes are identified - Operators and Administrators. Each user class has distinct roles and characteristics:

1. **Operators:**
    - **Frequency of Use:** The level of interaction with the monitoring system should be rather often and stable.
    - **Technical Expertise:** Possessing fundamental computer skills such as mouse control, keyboard operation, and touch screen navigation.
    - **Security Training**: Knowledge of some of the security measures concerning monitoring of environment and power supply.
    - **Access Levels:** Different levels of access in accordance with the user's role: viewing only the alerts and the results of the work, as well as setting the parameters of the system.
2. **Administrators**:
    - **Frequency of Use:** Occasional participation in system installation, upgrades, settings, and problem solving.
    - **Technical Expertise:** High level of technical competence in managing and maintaining the systems.

- **Security Oversight:** Monitors and audits the efficiency and quality of the monitoring system and the security personnel.
- **Access Levels:** Full control over the system configuration, maintenance, as well as audit options.

These user classes are important in the efficient running of the monitoring system. While the Operators are involved in the day-to-day monitoring activities, the Administrators are more involved in the maintenance of the system, monitoring, and constant enhancement.

3.1.4 Operating Environment

The SEMS operate in a versatile environment designed to accommodate both the monitoring hardware and the server component. Key elements of the operating environment include:

1. **Monitoring Hardware:**
    - Arduino Board
    - Sensors
    - Wireless Communication
2. **Server Component:**
    - Desktop Computer with the following minimum specification
        - Processor: Intel Core i5 or equivalent
        - Memory: 8 GB RAM or more
        - Storage: 100 GB HDD or more

- Operating System: Windows 10 or Linux

3. **Software Components:**
    - Arduino IDE
    - SMS Gateway Integration
    - Live Server
4. **Programming Framework:**
    - Arduino Programming Language
    - HTML, JavaScript, CSS
5. **External Devices:**
    - Display
    - SMS-Enabled GSM Module

This adaptable operating environment ensures the seamless integration of Arduino-basedmonitoring hardware and the server component, providing a robust platform for real-timeenvironmental and power supply monitoring.

3.1.5 Design and Implementation Constraints

SEMS is subject to specific design and implementation constraints that shape its development and functionality. These constraints include:

- **Legal and Ethical:** The monitoring system must adhere to local laws and regulations governing data privacy, environmental monitoring, and SMS

communications. The system should not violate public or individual privacy rights, and the collected data must not be misused.

- **Compatibility and Interoperability:** The monitoring system must seamlessly integrate with Arduino boards and sensors, requiring compatibility with Arduino IDE and communication protocols (Wi-Fi). The server component must be compatiblewith specified databases (MySQL) and software components for optimal performance.

- **Availability and Reliability:** The monitoring system must operate without internal or external causes leading to malfunctions, crashes, or freezes. This includes addressing bugs, errors, exceptions, power outages, network outages, or potential hostile attacks. Robust backup and recovery procedures must beimplemented to ensure data integrity and operational continuity.

- **Scalability and Maintainability:** The system must be scalable to handle diverse environmental data efficiently. Everything should be documented as to make it easier for future maintenance of the system.

3.1.6 Assumptions and Dependencies

The SEMS project relies on several key assumptions and dependencies:

- **Assumptions:** Assumes that environmental sensors and power supply monitors are appropriately placed and oriented for accurate data acquisition. Assumes the monitoring system operates effectively in environments where the monitored entities (e.g., sensors, Arduino components) remain relatively stationary during data acquisition. Assumes the availability of a consistent

and reliable power source for both the monitoring hardware (Arduino components) and the server component. Assumes the presence of a stable network connection for communication between monitoring systems installed at different locations and the central server database.

- **Dependencies:** The project is dependent on the Arduino board and associated sensors for environmental and power supply data acquisition. Dependency on a standard desktop computer to function as the server component, meeting specified hardware and software requirements. Dependencies on wireless communication protocols (e.g., Bluetooth, Wi-Fi) for seamless connectivity between Arduino components and the monitoring system.

3.2 External Interface Requirements

3.2.1 User Interfaces

The SEMS user interface prioritizes intuitive interaction, featuring:

- **Graphical User Interface (GUI):** Displays real-time data from environmental sensors and power supply monitors comprehensibly, showing temperature, humidity, air quality, and power supply status.
- **Standard Buttons and Functions:** Consistent presence of essential buttons like Settings, Alerts, and System Status, ensuring easy access to vital functions.
- **Intuitive Navigation:** Logical screen layout for smooth navigation, enhancing user experience for operators and administrators.

- **Error Message Displays:** Clear messages for prompt identification and resolution of system issues.
- **Responsive Design:** Compatibility with various display sizes and resolutions for flexibility across devices.

3.2.2 Hardware Interfaces

SEMS seamlessly interfaces with various hardware components to ensure efficient data acquisition and system functionality:

- **Arduino Board:** Interacts with attached sensors for real-time environmental and power supply monitoring.
- **Sensors:** Connects to a variety of sensors for comprehensive monitoring capabilities.
- **GSM Module:** Establishes communication for real-time SMS alerts in case of anomalies.
- **Server Component:** Communicates with a standard desktop computer for data processing and alert generation.
- **Communication Protocols:** Utilizes wireless protocols for seamless connectivity.
- **External Devices:** Interfaces with additional devices for visualization purposes

3.2.3 Software Interfaces

SEMS collaborates with various software components to ensure seamless functionality and efficient data processing:

- **Operating Systems:** Compatible with Windows, Linux, and macOS environments, ensuring accessibility across platforms.
- **Database:** Interfaces with MySQL for storing, retrieving, and managing environmental and power supply data.
- **Arduino Software:** Utilizes the Arduino development environment for programming and integrating Arduino boards into the system.
- **Communication Protocols:** Implements protocols for data exchange and SMS alerts via the integrated GSM module.
- **User Interface Framework:** Utilizes frameworks like React JS to design the graphical interface for users and administrators.
- **Arduino Libraries:** Incorporates specific libraries like Adafruit Sensor Library for seamless communication between Arduino boards and sensors

3.2.4 Communications Interfaces

SEMS relies on specific communication interfaces for seamless interactions and data exchange:

- **Wireless Connectivity:** Connects to Arduino boards and sensors by protocols such as Bluetooth or Wi-Fi to transmit information in real-time.

- **GSM Module:** Appeals to basic GSM framework for transmitting real-time alerts on exceptions of parameters in the system.
- **Server Communication:** Transmits information with the server part with using such protocols as HTTP or TCP/IP to deliver information and alerts.
- **Arduino Board Interface:** Connects to Arduino boards over the general-purpose communication interfaces provided by the Arduino integrated development environment.
- **Data Transfer Protocols:** Prescribes selected procedures by which the monitoring system can effectively exchange information with attached appliances.
- **Data Synchronization:** Ensures the use of synchronization mechanisms to keep data in the monitoring system and other relevant hardware in agreement.

3.3 System Features

3.3.1 Sensor Data Collection

3.3.1.1 Description and Priority

This feature consists of acquiring information from different sensors installed in the area of interest. It is of high priority since it serves as the basis for subsequent steps of analysis and decision-making.

3.3.1.2 Stimulus/Response Sequences

- **Stimulus:** Environmental sensors send real-time data.

- **Response:** The system captures, validates, and stores incoming sensor data.

3.3.1.3 Functional Requirements

The Sensor Data Collection of SEMS System has the following functional requirements along with their use case descriptions:

- **REQ-3.1.1:** The system should be able to incorporate with several sensors such as temperature, humidity, air quality, and motion sensors.

Use Case Name	Sensor Integration
Trigger	Integration request from a new sensor
Precondition	The system is operational
Basic Path	The system receives and verifies compatibility for new sensor integration requests before integrating them into the monitoring network.
Alternative Path	If the sensor type is incompatible, the system notifies the user and rejects the integration request
Postcondition	The new sensor is successfully integrated into the system.
Exception Path	Integration failure or incompatible sensor type.

- **REQ-3.1.2:** The system should provide real-time feedback on successful data capture and highlight any issues with sensor communication.

Use Case Name	Real-time Feedback
Trigger	Completion of sensor data capture
Precondition	Sensor data capture process is ongoing

Basic Path	The system captures sensor data, provides real-time feedback on successful data capture, and highlights communication issues for user attention.
Alternative Path	If communication issues persist, the system notifies the user of the problem.
Postcondition	User receives real-time feedback on data capture status.
Exception Path	Sensor communication issues detected.

3.3.2 Data Storage and Management

3.3.2.1 Description and Priority

This feature includes the safe deposit and easy organization of gathered information from the sensors. Thus, it is of high priority to provide and secure historical environmental data.

3.3.2.2 Stimulus/Response Sequences

- **Stimulus:** Validated sensor data is ready for storage.
- **Response:** The system stores data in the appropriate databases and performs necessary indexing for quick retrieval.

3.3.2.3 Functional Requirements

The Data Storage and Management of SEMS has the following functional requirements along with their use case descriptions:

- **REQ-3.2.1:** Data storage should be flexible in a way that it can handle large volumes of information as the storage advances.

Use Case Name	Scalable Data Storage

Trigger	Increasing volumes of information over time
Precondition	The system is operational
Basic Path	The system monitors stored information volume and scales up storage capacity if predefined thresholds are exceeded.
Alternative Path	If scaling fails, the system notifies the user and initiates error handling procedures.
Postcondition	The system accommodates increasing volumes of information with scalable data storage.
Exception Path	Scaling failure.

- **REQ-3.2.2:** For analytical requirements, the system should allow easy and fast data extraction.

Use Case Name	Efficient Data Retrieval
Trigger	User request for data analysis
Precondition	The system has stored data available
Basic Path	The user initiates a data analysis request, and the system efficiently retrieves the required data.
Alternative Path	If data retrieval encounters issues, the system notifies the user and initiates error handling procedures.
Postcondition	The user receives the required data efficiently for analysis.
Exception Path	Data retrieval failure.

3.3.3 Real-time Monitoring and Alerts

3.3.3.1 Description and Priority

This feature is useful to track the environmental values in real time and get alarms for the defined ranges. On this account, it is of high priority to make it possible to give timely responses to critical incidents.

3.3.3.2 Stimulus/Response Sequences

- **Stimulus:** Real-time data surpasses predefined thresholds.
- **Response:** The system triggers alerts and updates the real-time monitoring interface.

3.3.3.3 Functional Requirements

The Real-time Monitoring and Alerts of SEMS has the following functional requirements along with their use case descriptions:

- **REQ-3.3.1:** This is the list of functional requirements of the Real-time Monitoring and Alerts of SEMS along with their use cases.

Use Case Name	Customizable Alert Thresholds
Trigger	User defines or customizes alert thresholds
Precondition	The system is operational, and the user has appropriate permission
Basic Path	The user accesses the system's settings to define or customize alert thresholds based on environmental parameters.
Alternative Path	If threshold customization fails, the system notifies the user and provides feedback for resolution.
Postcondition	Customized alert thresholds are set for environmental parameters
Exception Path	Threshold customization failure.

- **REQ-3.3.2:** Notifications should be sent through several media such as SMS and Dashboard.

Use Case Name	Multi-Channel Alert Delivery
Trigger	System detects conditions exceeding defined thresholds
Precondition	The system is operational, and alert conditions are met
Basic Path	The system detects conditions exceeding user-defined thresholds and initiates alert delivery via SMS and dashboard.
Alternative Path	If alert delivery encounters issues, the system notifies the user and initiates error handling procedures.
Postcondition	Alerts are successfully delivered through multiple channels.
Exception Path	Alert delivery failure.

3.4 Other Non-Functional Requirements

3.4.1 Performance Requirements

- Process and respond to incoming sensor data in real-time with a maximum latency of 5 seconds.
- Generate SMS alerts within 10 seconds of detecting anomalies or failures.
- Handle concurrent requests from multiple users, maintaining a responsive user interface during peak usage.
- The delivery of SMS alert must have an uptime of 99% at the least to facilitate proper communication.

3.4.2 Safety Requirements

- The system must not interfere with activities of the facility or other systems such as electricity or HVAC.

- Sensors and other communication devices that are used in the hardware part of the system must meet safety standards and certifications.
- Integrate aspects that reduce the impacts of the system on other systems and the environment in the event of failure or abnormality.
- The user interfaces and the alerts that are to be incorporated into the design should be simple enough to comprehend especially at the time of the emergency.

3.4.3 Security Requirements

- To enhance the system security level, allow only the specific user identity to manipulate the GUI and settings, establish the strong password rules, and introduce the MFA.
- Introduce the standards such as GDPR to effectively manage the user data.
- Conduct scheduled security assessments and assessments for vulnerability with a view of addressing any gap that might be present in the system.

3.5 Software Quality Attributes

- **Adaptability:** The system should be flexible as well as capable of accommodating new sensors.
- **Availability:** The system should have high availability with minimal downtime for continuous monitoring.
- **Correctness:** Prioritize accurate data collection and reporting, promptly addressing any errors.

- **Flexibility:** Allow easy customization and configuration for different deployment scenarios.
- **Interoperability:** Seamless integration with various sensors, microcontrollers, and communication protocols.
- **Maintainability:** Ensure the codebase and components are well-documented for easy updates and modifications.
- **Portability:** Design the system to operate across different platforms and environments.
- **Reliability:** Consistently provide accurate and timely information, minimizing false alarms or missed events.
- **Reusability:** Design components and algorithms for reuse in future monitoring solutions.
- **Robustness:** Ensure resilience to power outages, network disruptions, or sensor malfunctions.
- **Testability:** Ensure that components undergo sufficient testing for proper validation and verification.
- **Usability:** Ensure that the application has an easy to use interface for the user regardless of their IT knowledge.

3.6 Business Rules

- The system should be able to detect the unauthorized objects with the accuracy of 98% and above for high security.

- The system has to be fully independent, perform the monitoring without requiring constant human interference, and transfer the data and the alerts through an HTTPS interface.

- Make sure the system does not violate any regulations concerning its design and functioning, thus not affecting any necessary certifications or approvals.

3.7 Other Requirements

- Facilitate storage and retrieval of data from a centralized large database that can accommodate expected environmental data.

- Interface with other environmental monitoring systems and structures, in this way compatible with different sensors and data.

- Cater for the users who come from different linguistic background by translating the user interface, the alerts and the system messages into different languages.

Chapter 4: System Architecture & Design

4.1 Architectural Design

The design of SEMS entails the use of the client-server model to enhance the monitoring and management of the system. Below, the system's architecture has been illustrated:

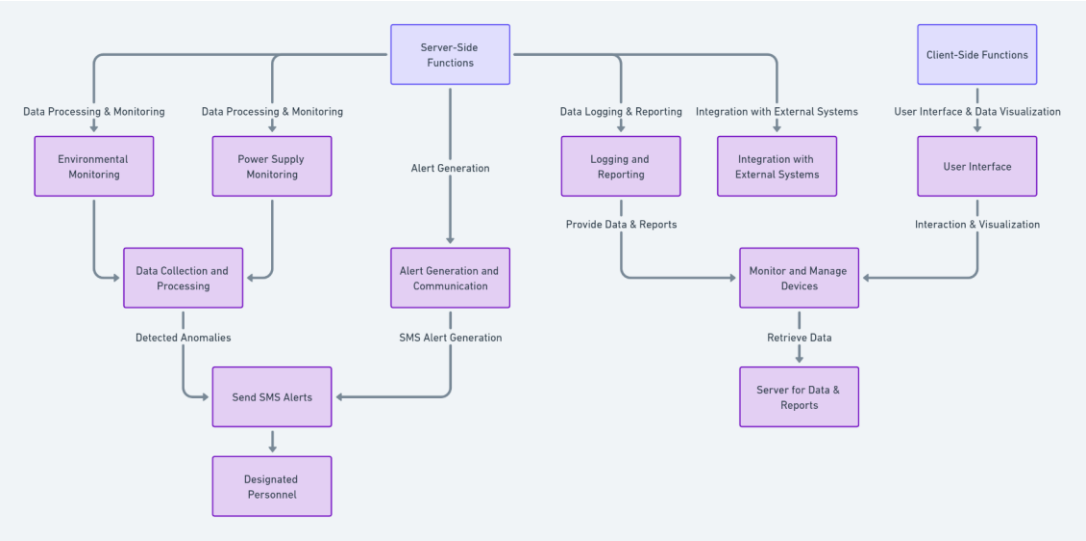

Figure 14 - Architecture Diagram

Client-Server Architecture: SEMS adopts a client-server architecture, which is well-suited for ensuring robust and efficient operation. This design enhances scalability, flexibility, and real-time monitoring capabilities.

The selection of a client-server architecture is driven by the following considerations:

- **Modularity:** This means that the client-server model maintains a clear distinction between responsibilities. It splits the system into clear client-side and server-side layers which helps

in independent development and testing. This modularity makes the system easy to manage and maintain.

- **Maintainability:** Separating the graphical user interface (frontend) from the data processing and control structures (backend) guarantees that modifications or enhancements in one part of the structure cannot majorly affect another. This maintainability is important, especially when the monitoring requirements are likely to change in the future.
- **Scalability:** The client-server architecture is easy to scale since it supports client-server relationships on both the client and server sides. New monitoring features, extra sensors, or even modifications of the graphical user interface can be easily implemented without affecting the whole system. This scalability enables SEMS to effectively add more functions to it in order to meet the future environmental and power supply monitoring requirements.

4.2 Use-case Diagram

The various specific features of SEMS are described in detail through the use cases that are at the core of this model. This section provides a flowchart that illustrates the relationships of the system's users with its parts, as well as the process of observing environmental conditions and power supply status. It goes to the extent of describing factors which prompt system alerts, circumstances under which the alerts are raised, and the anticipated results of the interaction. The use case diagram together with its description is useful in providing a road-map on how the system should behave in the real world which is crucial in the understanding of the system by the technical and non-technical persons. The SEMS is to be used for constant and effective surveillance, response to alerts, and flexible system configuration for the best operating capacity. Following is the Use-case Diagram of SEMS.

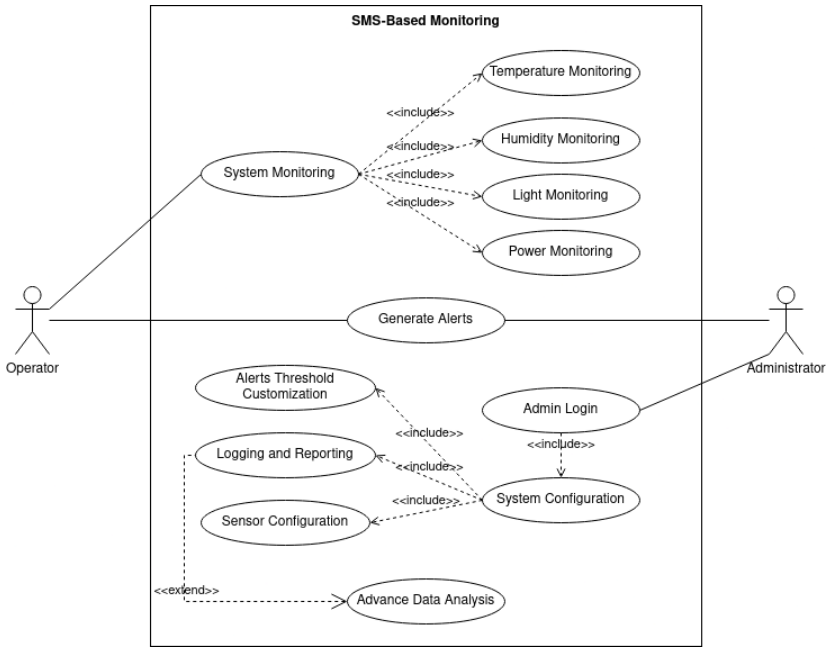

Figure 15 - Use-Case Diagram

4.3 Activity Diagram

The activity diagram represents the workflow of SEMS. It unfolds as follows:

- **Start:** The process is initiated, activating the monitoring system.
- **Login:** An authorization step where the user logs into the system to access the monitoring features.
- **Admin Credential?** A decision node that routes the user to either the Admin Dashboard or User Dashboard based on their credentials.
- **Admin Dashboard/User Dashboard**: Depending on the credentials provided, the user is directed to the appropriate dashboard. An admin can configure system settings, while a general user can view system status.

- **System Configuration:** For admins, this is where they can set up thresholds and parameters for monitoring environmental factors and power supply conditions.
- **Configuration Completed?** A decision node checks if the system configuration is complete.
- **System Monitoring:** For users, continuous monitoring of environmental parameters and power supply status occurs here. If certain predefined conditions are met or thresholds are crossed, the system proceeds to the next step.
- **Monitoring Completed?** A decision node that checks if monitoring is ongoing or if it should proceed to generate alerts.
- **Generate Alerts:** If the monitoring system detects anomalies or any conditions that require attention, it generates alerts. Once alerts are generated, they can be sent out as SMS to the relevant stakeholders or personnel for immediate action.
- **Alerts Generation Complete?** A decision node that confirms whether the alert generation process is complete.
- **Logout:** User session is closed.
- **End:** The process has terminated, closing the monitoring system.

Throughout the process, decision nodes guide the workflow, determining whether to proceed with configuration, monitoring, alert generation, or to end the session after logging out. This systematic

approach ensures that environmental and power supply monitoring is both efficient and responsive to the conditions being monitored. Below is the Activity Diagram for SEMS.

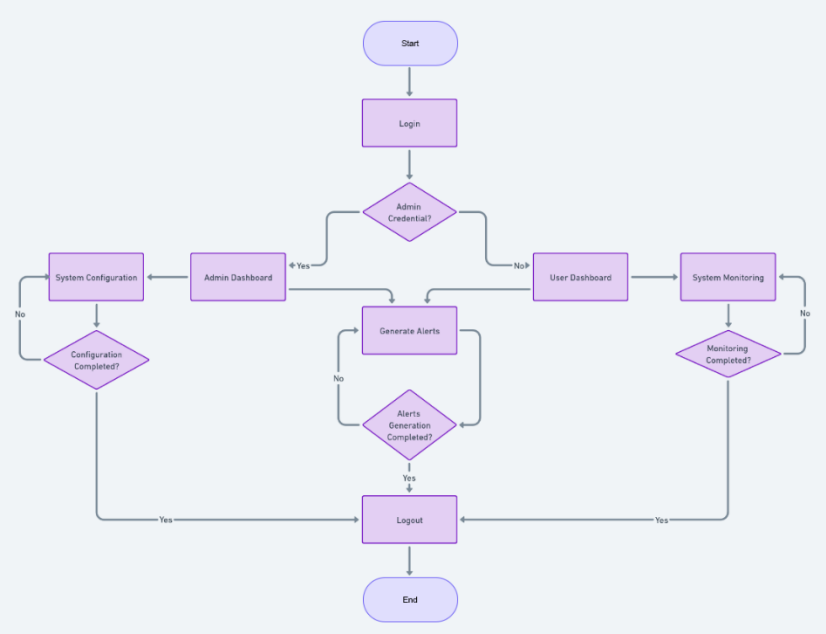

Figure 16 - Activity Diagram

4.4 Sequence Diagram

The sequence diagram delineates the series of interactions among different components of SEMS, each represented by a separate point:

- **User:** Begins the sequence by accessing the system through the User Interface. The User's role is to interact with the system to initiate monitoring.
- **Administrator**: Accesses the system for administrative functions, possibly including configuration and system oversight, indicating a higher level of control and system access.
- **Login:** Acts as the gateway for both the User and the Administrator, verifying credentials and granting access to the system.

- **Server:** The Server is a critical node that processes requests from the User Interface, obtains and processes data from the Sensors, and sends configuration updates. It represents the processing and logical decision-making center of the system.
- **Sensors:** Play a vital role by collecting environmental data and sending it to the Server for analysis.
- **Database:** Database is involved in storing the data collected by the Sensors and processed by the Server.
- **GSM Module**: Functions as the alert mechanism by sending SMS alerts to designated personnel, as triggered by the Server based on specific criteria or thresholds.
- **Personnel**: Represent the end receivers of the system's output, responsible for responding to alerts and taking appropriate actions based on the SMS messages received.

The sequence clearly depicts the flow of data and control from the user's initial interaction through to the final alert stage, with each actor playing a distinct and necessary role in the system's operation. Below is the Sequence Diagram for SEMS.

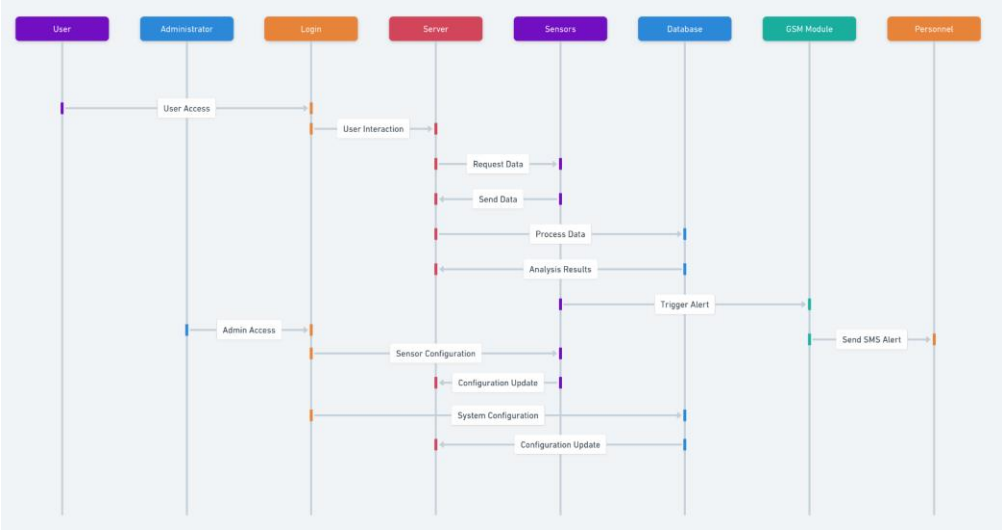

Figure 17 - Sequence Diagram

4.5 Data Flow Diagram

The SEMS (SMS-Based Environment and Power Supply Monitoring System) utilizes a detailed data structure to support its operations and enhance its effectiveness. This section focuses on the data descriptions and dictionary, offering insights into the information processed by the system. Below is the corresponding Data-flow Diagram:

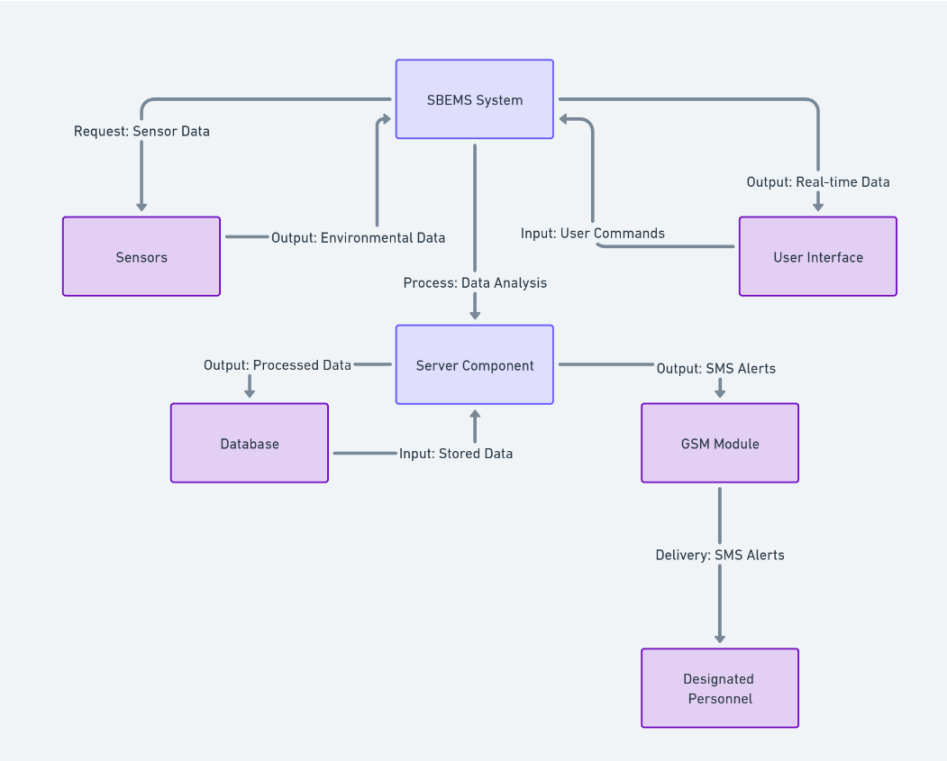

Figure 18 - Data-Flow Diagram

Chapter 5: Hardware Design

The hardware design plays an important role of the SMS-based environment monitoring system since it contains the basic data acquisition, processing and transmitting system. This chapter begins a detailed exploration of the various aspects of the hardware design, including circuit diagrams, the components used in the nodes and the approaches used in integrating them across the various nodes.

5.1 Hardware Architecture Overview

As the basis of the monitoring system, a carefully designed hardware structure is the key to the effective integration of various components. The architecture includes the main receiver node and different slave nodes that are designed to monitor different environmental variables. These nodes work in harmony to collect, analyze, and transmit important information to the master database server. The architecture itself must be highly reliable and scalable to accommodate the changing demands of monitoring as well as environmental factors.

5.2 Main Receiver Node

5.2.1 Components

Main receiver node is the core of the entire monitoring system where data from different sources are collected and forwarded to the database server. It comprises several integral components:

- **Arduino Microcontroller:** The Arduino board is the central processing unit of the system and is responsible for initiating communication with the slave nodes and data transfer. Because of this, it is perfect for controlling the

general functioning of the system due to its flexibility and simple programming.

- **Network Shield:** This shield offers Ethernet connection ensuring that the main receiver node connects to the local network and the internet for efficient data transfer. The shield incorporates an RJ45 Ethernet jack that enhances the data transfer rate and maintains a steady network connection.

- **ESP32 Module:** When connected to the Arduino board, the ESP32 module provides the wire-less interface which allows communication with the slave nodes through Wi-Fi. It has got a dual-core processor and support for Bluetooth Low Energy (BLE) which makes it a very efficient chip for wireless data transfer.

- **GSM Module (SIM900):** The GSM module improves the system's performance by allowing the sending of SMS alerts and notifications on the occurrence of critical events in the environment. Due to its compatibility with the standard SIM card and support of various communication protocols, it can be used for various remote monitoring applications.

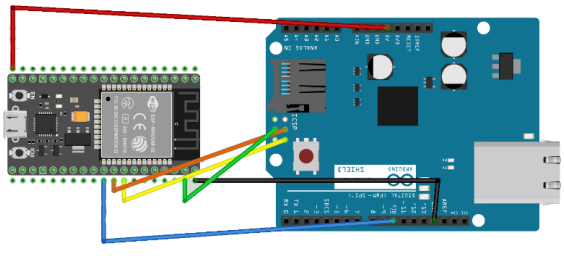

Figure 19 - Schematic Diagram of Main Receiver Node - 1

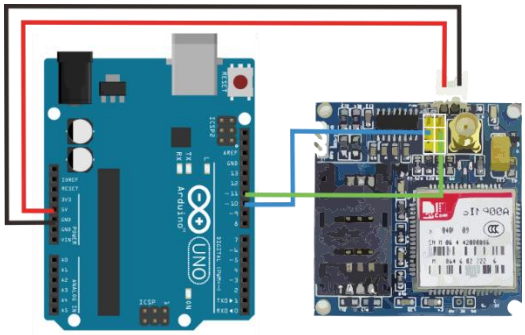

Figure 20 - Schematic Diagram of Main Receiver Node - 2

5.2.2 Code Explanation

In the ESP32 code, the wireless communication between the sensor nodes and the main receiver node is enabled. It starts by creating a structure to take data, which corresponds to the structure of the sender and a callback function (`OnDataRecv`) to deal with the received data. This function provides the update of stored sensor values and triggers an SMS alert should a threshold be attained. The following highlights key aspects of the code:

- **Data Reception:** The `'OnDataRecv'` function is called when data is received from the sensors, and sets the temperature, humidity, air quality, voltage, and current values. For example, it checks and prints the temperature value:

```
Serial.print("temperature is:"); Serial.println(temp);
```

- **Threshold Checks and SMS Alerts:** The system checks whether the received values are beyond the set limit. If so, it invokes functions such as `'send_temp_sms()'` to send alert messages through the GSM module. For instance, when the temperature exceeds the threshold:

```
if (temp > temp_thr) { send_temp_sms(); }
```

- **Data Transmission to Arduino:** Said values are refreshed and sent to the Arduino through the `'Serial.write'` method, which allows the Arduino to obtain updated data for further computation and entry into the database. For instance, sending the temperature data:

```
Serial.write((uint8_t*)&temp, sizeof(temp));
```

The Ethernet control and management of communication with the ESP32 are implemented in the Arduino code. It is designed to read sensor data, analyze it, and then send it to a server for database storing. Key points include:

- **Ethernet Setup:** This step is used to connect the Ethernet shield with the network.

```
if (Ethernet.begin(mac) == 0) { Serial.println("Failed
to configure Ethernet using DHCP"); }
```

- **Data Reception:** The Arduino constantly receives sensor data from the ESP32 so that there is always a match in terms of environmental readings.

```
while (esp.read() != '#'); // Wait until start of frame
marker is received
```

- **HTTP Request for Data Logging:** The `'Sending_To_phpmyadmindatabase'` function constructs the HTTP string and sends the HTTP request to the remote database to log the sensor data for data integrity and easy access.

Thus, these elements combined, help the main receiver node accumulate and broadcast environmental data, which are capable of giving near-real-time readings of important environmental data points.

5.3 Temperature/Humidity Slave Node

5.3.1 Components

The Temperature/Humidity Slave Node is the node that measures the environmental conditions, namely temperature and humidity. It includes the following components:

- **DHT11 Sensor:** This sensor measures temperature and humidity, and gives output in digital format, which makes it easy to acquire data.
- **ESP32 Module:** The ESP32 is in charge of the wireless communication with the main receiver node and guarantees the proper data transmission.
- **18650 Battery:** This battery offers mobility and can be placed anywhere near the node hence offering flexibility.
- **Button:** It is used for manual on/off operations and reset functions.

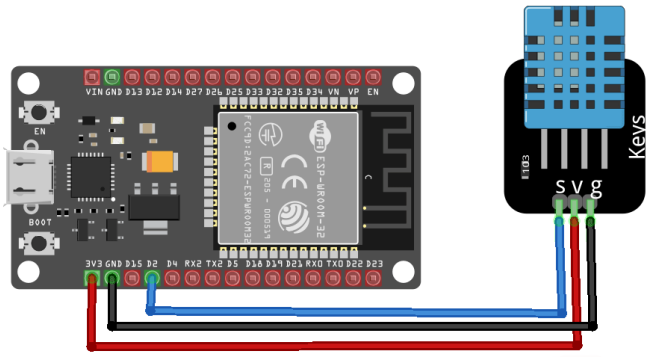

Figure 21 - Schematic Diagram of Temperature/Humidity Slave Node

5.3.2 Code Explanation

The code begins ESP-NOW connection and configuration of the DHT11 sensor to read the temperature and humidity data. The values are then transmitted to the main receiver node through ESP-NOW.

```
// Declare Variables
float humid = dht.readHumidity();
float temp = dht.readTemperature();

// Set values to send
myData.id = 1;
myData.x = temp;
myData.y = humid;

// Send message via ESP-NOW
esp_err_t result = esp_now_send(broadcastAddress,
(uint8_t *) &myData, sizeof(myData));
```

5.4 Air Quality Slave Node

5.4.1 Components

The AQ slave node is tasked with the role of sensing different pollutants that are found in the air such as CO2, NH3, and VOCs. It includes the following components:

- **MQ135 Sensor:** This sensor has the capability of measuring air quality through identification of these pollutants. Analog output is fed to the ESP32 microcontroller for further processing.
- **ESP32 Module:** The ESP32 is in charge of the wireless communication with the main receiver node and guarantees the proper data transmission.
- **18650 Battery:** This battery offers mobility and can be placed anywhere near the node hence offering flexibility.

- **Button:** It is used for manual on/off operations and reset functions.

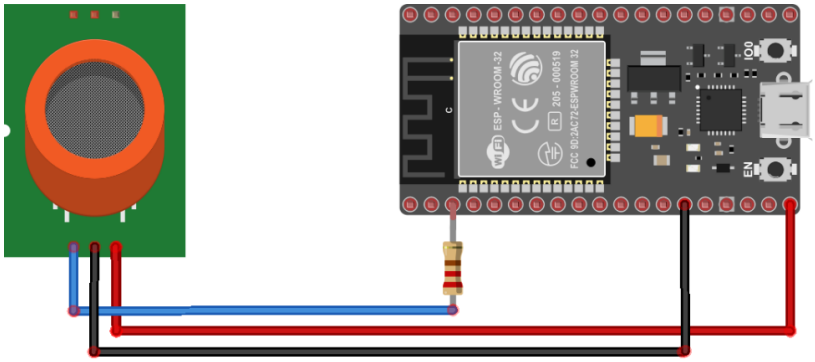

Figure 22 - Schematic Diagram of Air Quality Slave Node

5.4.2 Code Explanation

The code starts up the ESP-NOW communication and configures the MQ135 sensor for measuring the air quality index. The sensor values are then transmitted to the main receiver node using ESP-NOW.

```
// Declare Variables
float airqual = analogRead(mq135);

// Set values to send
myData.id = 2;
myData.x = airqual;

// Send message via ESP-NOW
esp_err_t result = esp_now_send(broadcastAddress,
(uint8_t *) &myData, sizeof(myData));
```

5.5 Volt/Current Slave Node

5.5.1 Components

The Volt/Current Slave Node measures the voltage and current, providing information about the power and electrical conductivity. It includes the following components:

- **ZMCT103C Sensor:** This sensor is used for measuring voltage and current. The ESP32 microcontroller processes its analog output.
- **ESP32 Module:** The ESP32 handles the wireless communication with the main receiver node, ensuring reliable data transmission.
- **Capacitors:** This component stabilize the sensor readings and help filter out noise.
- **External Power Bank:** This power source allows for flexible placement and continuous operation of the node.
- **Button Switch:** Used for manual operations and reset functions.

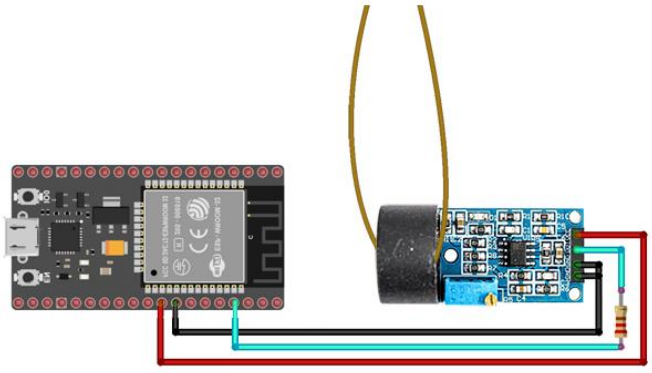

Figure 23 - Schematic Diagram of Volt/Current Slave Node

5.5.2 Code Explanation

The code initializes the ESP-NOW protocol and sets up the ZMCT103C sensor for reading voltage and current values. The values are then sent to the main receiver node using ESP-NOW.

```
// Read the sensor value
float sensorValue = analogRead(voltPin);
float voltage = (sensorValue / 4096.0) * 300.0;
float vpp = computeVPP();
float peakCurrent = (vpp / 110.0) * 1000.0;
float rmsCurrent = peakCurrent * 0.707;

// Prepare the data to send
myData.id = 3;
myData.x = voltage;
myData.y = (rmsCurrent * 100);

// Send data using ESP-NOW
esp_err_t sendStatus = esp_now_send(broadcastAddress,
(uint8_t *)&myData, sizeof(myData));
```

5.6 Hardware Architecture Overview

Wireless communication channels, established by the ESP32 modules, facilitate seamless data exchange be-tween the main receiver node and slave nodes. Upon receiving data from the slave nodes, the main receiver node processes and aggregates the information before transmitting it to the database server via the Ethernet connection provided by the Network Shield. The use of standard communication protocols ensures compatibil-ity and interoperability between different hardware components, enabling efficient data transmission and system operation.

5.7 Power Management and Efficiency

Efficient power management strategies are employed to ensure the longevity and reliability of the monitoring system. Each node is equipped with a rechargeable 18650 battery, providing a reliable power source for ex-tended operation. Additionally, low-power components and sleep modes in the ESP32 microcontrollers mini-mize power consumption, optimizing energy efficiency and prolonging battery life. The use of external power banks ensures uninterrupted power supply to the volt/current slave node, allowing it to monitor electrical pa-rameters without relying solely on battery power.

5.8 Scalability and Expansion

The hardware design of the monitoring system is also scalable, which means that it can be added more sensors or functionalities easily to the system. As the monitoring requirements evolve over time, it is easy to add new sensor modules or peripheral devices using the modular design principles. Available expansion ports and general purpose input/output on both Arduino and ESP32 boards offer numerous interfaces for including extra sensors, controls, or communication interfaces, thus making the system future-proof for any changes in environmental monitoring requirements.

5.9 Testing and Validation

In deployment, hardware devices go through series of functional, reliability and accuracy tests in order to make sure that it performs the intended role. It is for this reason that intense performance testing under various environmental conditions, testing for data ad-equacy and coherence and testing of the communication pro-tocols are critical measures in the effectiveness of the monitoring system. Application and field trials help to determine effectiveness of the system in various work conditions, which will be important to understand when applying various adjustments.

Chapter 6: Software Design

The software design of the SMS Based Environment Monitoring System is made up of an elaborate system architecture that brings out aspects relating to data processing, user –system interface, and the overall communication that exists between the front end and the back end of the system. This chapter focuses on the desired look and functionality of both the interface presented to the user and the functionality behind the scenes, the technologies employed and the database design.

6.1 Concept

The selection of new generation web technologies for the UI design of the SEMS was done to implement the simplistic, easier to access and real-time data representation. Using HTML, CSS, JavaScript and even some of the more complex frameworks like React, the SEMS UI enables users to track environmental variables, through SMS.

The idea of adopting the component-based architecture and dependence on the React, makes it possible for the SEMS UI to be scalable, maintainable as well as easy to develop. Being a multifunctional device, the UI of the device is made in such a way that it can monitor and control various measures like the temperature, humidity, air quality, and voltage or current levels to help the users make the right decisions.

6.2 Frontend Design

The frontend of the SEMS corresponds to the user interface (UI) of the system through which the users engage with the software program. It has an elegant design, it is interactive and, in general, it has been carefully developed so that the flow is as smooth as possible. The frontend side is developed as a JavaScript application that is a single-page application (SPA) based on the React

framework, which enables the smooth user interactions without the necessity of frequent page reloads.

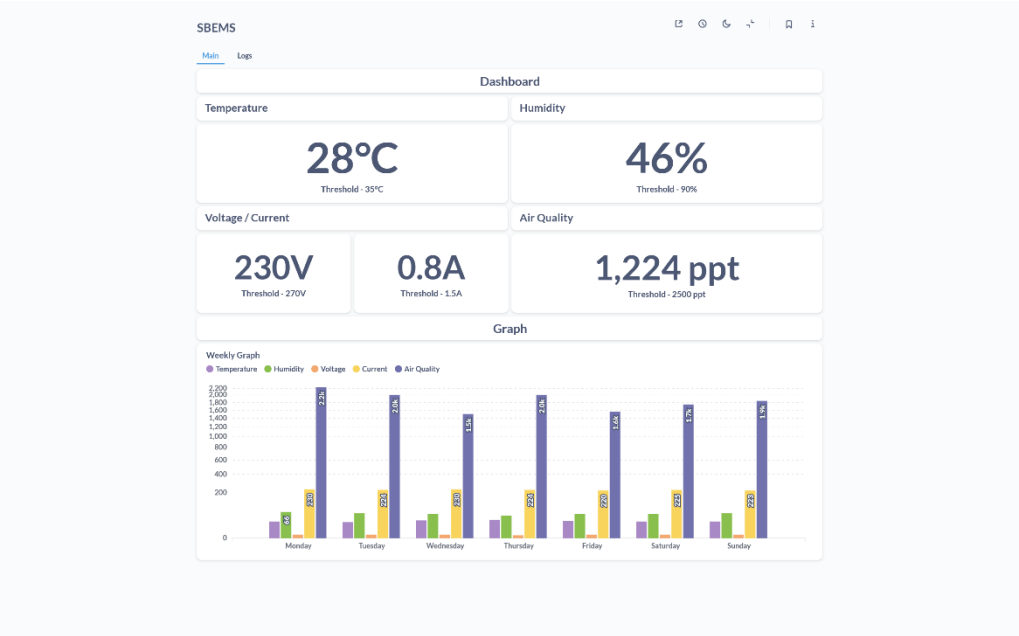

Figure 24 - SEMS Interface

6.2.1 Technologies Used

- **HTML, CSS, and JavaScript:** Core web technologies used for structuring, styling, and adding interactivity to web pages
- **React Framework:** Employed for building reusable UI components and managing the application state efficiently.
- **TypeScript:** Provides static typing to JavaScript, enhancing code quality and maintainability.
- **ClojureScript:** A variant of Clojure that compiles to JavaScript, used for its functional programming paradigms and compatibility with the rest of the stack.

- **Webpack:** A module bundler that compiles JavaScript modules, ensuring efficient loading and optimization of resources.
- **Babel:** A JavaScript compiler that converts modern JavaScript (ES6+) into a version compatible with older browsers.
- **cssnext:** A tool that allows the use of future CSS syntax today, enabling modern styling practices.

6.2.2 Frontend Architecture

The frontend follows a component-based architecture, where each UI element is encapsulated within a React component. These components are organized hierarchically to build complex UIs from simple, reusable elements.

- **UI Components:** Reusable pieces of the interface, such as buttons, forms, and tables, implemented as React component.
- **State Management:** Managed using React's Context API and hooks, ensuring a centralized and consistent application state.
- **Routing:** Implemented using React Router to manage navigation within the SPA, allowing for smooth transitions between different views.

6.2.3 Frontend Features

The SEMS frontend is designed with user convenience and real-time monitoring in mind. The interface includes two main tabs: a default dashboard tab and a logs tab. Each tab is designed to provide specific functionalities to the user.

- **Dashboard Tab**: Displays real-time data for temperature, humidity, air quality, voltage, and current. It includes:

    1. **Field Containers:** Each parameter (temperature, humidity, air quality, voltage, current) has its own container that displays real-time data.

    2. **Weekly Graph:** Displays average values for each parameter per day, providing a historical view of environmental conditions.

    3. **Automatic Refresh:** Allows users to set a refresh timer from a dropdown menu, refreshing only the data containers without reloading the entire page.

    4. **Dark Mode Toggle:** Users can switch between light and dark modes, enhancing usability in different lighting conditions.

- **Logs Tab:** Provides a paginated view of the complete database, displaying 10 rows per page. This allows users to browse historical data efficiently. The logs tab includes**:**

    1. **Pagination Controls:** Go to each of the pages in the database logs.

    2. **Search Functionality:** Search for specific functions of the Database or entries of the Database.

    3. **Sorting:** Sort the data based on different columns for analysis in a more convenient manner.

Figure 25 – Dark Mode

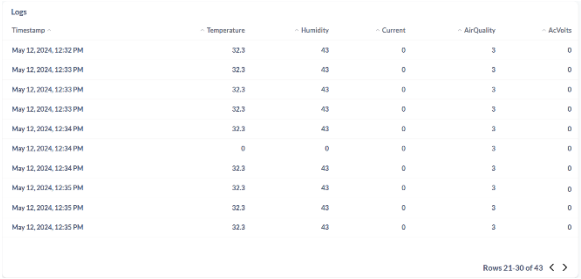

Figure 26 – Log Tab

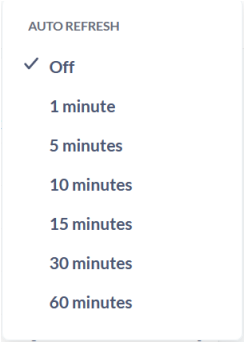

Figure 27 - Auto-Refresh

6.3 Backend Design

In the architecture of the SEMS, the backend deals with computation, application computational operations, and data persistence using the database. The backend is designed with Apache Server (2.4.58) and connected to MariaDB (10.4.32) database, using popular functional language Clojure for database access and back-end part.

6.3.1 Technologies Used

- **Apache Server:** The web server has the primary responsibilities to manage the HTTP recurrent and to deliver static and/or dynamic stages.
- **MariaDB (MySQL):** The architecture of an RDBMS for storing information collected by sensors, user's data and configurations of the system.

- **Clojure:** A programming language that is now being used in writing server-side application and dealing with the database. Due to the focus on the concepts of immutability and concurrency Clojure can be successfully used to develop reliable backend systems.

6.3.2 Backend Architecture

The implementation of the backend is divided into layers in order to improve separability and flexibility.

- **API Layer:** Creates RESTful endpoints to be used by the front end to communicate with the back end which is done using Clojure web frameworks.
- **Service Layer:** Stores business logic and manipulates the data that comes in or goes out of the database.
- **Data Access Layer:** Works with the MariaDB database and updates it by the conventions of CRUD (Create, Read, Update and Delete).

6.3.3 Database Design

The database schema is designed to efficiently store and manage data related to environmental monitoring, user management, and system configurations. The database named `'monitoring'` contains two key tables: `'smsbm'` and `'weekly'`.

- **'smsbm' Table**:
    - `'Timestamp'` (timestamp): Records the exact time of each sensor reading.

- `'temperature'` (float): Stores temperature readings.
- `'humidity'` (float): Stores humidity readings.
- `'current'` (float): Stores current measurements.
- `'airQuality'` (float): Stores air quality readings.
- `'acVolts'` (float): Stores AC voltage measurements.

```
CREATE TABLE smsbm (
    Timestamp TIMESTAMP,
    temperature FLOAT,
    humidity FLOAT,
    current FLOAT,
    airQuality FLOAT,
    acVolts FLOAT
);
```

Timestamp	temperature	humidity	current	airQuality	acVolts
2024-05-12 12:35:13	32.3	43	0	3	2
2024-05-12 12:35:29	32.3	43	0	3	0
2024-05-12 12:35:45	32.3	43	0	3	0
2024-05-12 12:36:33	32.3	43	0	0	0
2024-05-12 12:36:51	32.3	43	0	0	0

Figure 28 – 'smsbm' Table

- **'weekly' Table**:
  - `'Datetime'` (datetime): Records the exact date and time of each avg reading update.
  - `'day'` (varchar(11)): Records the day of the week.
  - `'avgtemp'` (float): Stores the average temperature for the day.

- `'avghumid'` (float): Stores the average humidity for the day.
- `'avgcurr'` (float): Stores the average current for the day.
- `'avgairqual'` (float): Stores the average air quality for the day.
- `'avgacvolt'` (float): Stores the average AC voltage for the day.

```
CREATE TABLE weekly (
    Datetime DATETIME,
    day VARCHAR(11),
    avgtemp FLOAT,
    avghumid FLOAT,
    avgcurr FLOAT,
    avgairqual FLOAT,
    avgacvolt FLOAT
);
```

Datetime	day	avgtemp	avghumid	avgcurr	avgairqual	avgacvolt
2024-05-19 14:59:07	Monday	26	66	230	2230	0.89
2024-05-19 14:59:26	Tuesday	24	60	224	2000	0.93
2024-05-19 14:59:43	Wednesday	30	56	230	1500	0.95
2024-05-19 15:00:03	Thursday	32	50	224	2000	0.8
2024-05-19 15:00:24	Friday	28	56	220	1560	0.9
2024-05-19 15:00:40	Saturday	26	58	225	1740	0.91
2024-05-19 15:02:54	Sunday	26	60	223	1850	0.95

Figure 29 – 'weekly' Table

- **Event Scheduler**:

To automate the aggregation of daily sensor data and update the weekly averages, an event scheduler task is employed. This task runs once per day, calculates the daily averages, and inserts them into the `'weekly'` table.

```
ON SCHEDULE EVERY 1 DAY STARTS '2024-05-08 00:00:00'
ON COMPLETION NOT PRESERVE ENABLE
DO BEGIN
    DECLARE current_day VARCHAR(11);
    DECLARE avg_temp FLOAT;
    DECLARE avg_humidity FLOAT;
    DECLARE avg_current FLOAT;
    DECLARE avg_air_quality FLOAT;
    DECLARE avg_ac_volts FLOAT;

    -- Get the current day
    SET current_day = DAYNAME(CURRENT_DATE);

    -- Calculate averages for the current day
    SELECT
        AVG(temperature),
        AVG(humidity),
        AVG(current),
        AVG(airQuality),
        AVG(acVolts)
    INTO
        avg_temp, avg_humidity, avg_current, avg_air_quality,
avg_ac_volts
    FROM monitoring.smsbm;

    -- Insert averages into the weekly table
    INSERT INTO monitoring.weekly (day, avgtemp, avghumid,
avgcurr, avgairqual, avgacvolt)
    VALUES (current_day, avg_temp, avg_humidity, avg_current,
avg_air_quality, avg_ac_volts);
END;
```

Chapter 7: Conclusion

This thesis has proposed an idea of an environment monitoring system, which is referred to as SEMS, which clearly marks a paradigm shift from manual collection of data to semi-automated/ real-time monitoring and alert systems. The strategies employed in the SEMS centered on superior sensor technology and wireless communications, combined with robust SMS notification to improve the effectiveness and efficiency of environmental monitoring.

The aims and objectives of the SEMS are successfully met by providing accurate and immediate measurement of environment parameters like temperature, humidity, air quality, voltage, and current. With the help of sensors such as DHT11, MQ135 and ZMCT103C, the system can gather accurate data while transmitting it properly to the main receiver node. The data is then compiled and assessed to create awareness of critical changes in the environment through the delivery of automatic alerts in form SMS.

In addition to being efficient, the design of the SEMS also outperforms the reliability of the usual monitoring techniques. It is a solution developed to increase the effectiveness of environmental surveillance while being substantially cheaper than some other systems used in other places. This greatly makes it possible for the system to be implemented in almost any context inclusive of industrial use, agriculture, and homesteads.

In other words, the SEMS embodies the union of two highly potent aspects of the contemporary context, namely technological advancement and strategic planning, and is poised to become an incredibly beneficial tool in the field of environment management.

Chapter 8: Future Work

Future milestones that need to be achieved to commercialize the SEMS are the following:

8.1 Integration with IoT Platforms

Extending the implementation of the SEMS to interconnect with other well-known IoT solutions like AWS IoT, Microsoft Azure IoT, or Google Cloud IoT could improve it. Such platforms are helpful as they also provide machine learning features and sophisticated analytics as well as data storage capabilities. From these platforms, SEMS can give more detailed two-way analysis and forecast on trends rather than just the aftermaths; hence, allowing for more preventive or preemptive measures.

8.2 Advanced Data Analytics and Machine Learning

The use of the most recent statistical and computational tools such as data analytics and the use of machine learning can greatly complement the SEMS. For instance, in the use of predictive models one can predict changes in the environment from past data, and take necessary action beforehand. The collected data can also be analyzed using machine learning algorithms, and based on the results, potential problems will be detected in which attention must be paid immediately.

8.3 Enhanced Communication Capabilities

To make the information transfer secure in any environment, the SEMS can incorporate multiple numbers of communication interfaces like the LoRaWAN, NB-IoT and Sigfox also for the existing unit other than SMS. Through concurrent use of the multiple channels, information can be relayed even in areas with low signals thus improving on the reliability of the system.

8.4 National and International Standards

To ensure compliance to required standards for quality management or conformity to the European market, for example CE marking, conformity to some degree with the ISO 9001 standard, will be prerequisites for commercial implementation. It includes several sites reviews, sign-offs and checks to ascertain that SEMS fulfills all the legal standards and performs in an optimized manner under different circumstances.

In this context, the integration of these future milestones can help establish the SEMS as a highly flexible, scientifically sound and commercially attractive solution to the monitoring of the environment. These advancements, however, will improve its current functioning and pave the way for new areas of use in various fields, which will lead to the enhancement of environmental management and protection.

References and Work Cited

1. IEEE. *IEEE Recommended Practice for Software Requirements Specifications*. IEEE Std 830-1998, 1998. [online] Available at: https://ieeexplore.ieee.org/document/720574

2. International Organization for Standardization (ISO). *Environmental Monitoring Standards*. [online] Available at: http://www.iso.org/files/live/sites/isoorg/files/archive/pdf/en/theiso14000family_2009.pdf

3. GSM Association. *GSM Module Integration Guidelines*. [online] Available at: https://www.gsma.com/futurenetworks/wp-content/uploads/2022/04/GSMA-Operator-Platform-Telco-Edge-Requirements-2022-v2.0.pdf

4. Chen, Z., Deng, G., & Wu, H. (2008). *A GSM-based remote wireless automatic monitoring system for field information: A case study for ecological monitoring of the oriental fruit fly, Bactrocera dorsalis (Hendel)*. [online] Available at: https://www.sciencedirect.com/science/article/abs/pii/S0168169908000148

5. Pran, K., Sahu, O. P., & Mohapatra, S. (2018). *Wireless Sensor Network for Monitoring & Control of Environmental Factors using Arduino*. [online] Available at: https://www.researchgate.net/publication/324099742_Wireless_Sensor_Network_for_Monitoring_Control_of_Environmental_Factors_using_Arduino

6. Arduino. *Arduino UNO R3*. [online] Available at: https://docs.arduino.cc/hardware/uno-rev3/

7. Arduino. *Ethernet Shield Rev2*. [online] Available at: https://docs.arduino.cc/hardware/ethernet-shield-rev2/

8. Espressif. *ESP-IDF Programming Guide*. [online] Available at: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/index.html

9. SIMCom. *SIM900 Hardware Design*. [online] Available at: https://simcom.ee/documents/SIM900/SIM900_Hardware%20Design_V2.05.pdf

10. Mouser Electronics. *DHT11 Humidity & Temperature Sensor*. [online] Available at: https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf

11. Winsen. *MQ135 Air Quality Gas Sensor*. [online] Available at: https://www.winsen-sensor.com/d/files/PDF/Semiconductor%20Gas%20Sensor/MQ135%20(Ver1.4)%20-%20Manual.pdf

12. Micro Transformer. *ZMCT103C Voltage/Current Sensor*. [online] Available at: https://www.micro-transformer.com/pcb-mounting-current-transformer-ZMCT103C.html

13. Clojure Documentation. *Clojure Guides*. [online] Available at: https://clojure-doc.org/

14. Apache Friends. *XAMPP Documentation*. [online] Available at: https://www.apachefriends.org/docs/

15. Mozilla Contributors. *MDN Web Docs - HTML: HyperText Markup Language*. [online] Available at: https://developer.mozilla.org/en-US/docs/Web/HTML

16. Mozilla Contributors. *MDN Web Docs - JavaScript*. [online] Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript

17. React. *React Documentation: A JavaScript library for building user interfaces*. [online] Available at: https://reactjs.org/docs/getting-started.html

Annexures

Anx-A

Hardware Code

- **Arduino**

```
#include <SoftwareSerial.h>
SoftwareSerial espSerial(2, 3);

#include <SPI.h>
#include <Ethernet.h>

byte macAddress[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED }; // MAC Address

char serverAddress[] = "xxx.xxx.xxx.xxx";
IPAddress localIp(xxx, xxx, xxx, xxx);
EthernetClient netClient;

float temperature;
float humidity;
float airQuality;
float voltage;
float current;

void setup() {
  espSerial.begin(9600);
  Serial.begin(9600);

  if (Ethernet.begin(macAddress) == 0) {
    Serial.println("Failed to configure Ethernet using DHCP");
    Ethernet.begin(macAddress, localIp);
  } else {
    Serial.println("Connected Successfully");
  }
  delay(1000);
}

void loop() {
  if (espSerial.available() >= 5 * sizeof(float) + 2 * sizeof(int)) {
```

```
    // Wait until start of frame marker is received
    while (espSerial.read() != '#');
    delay(10);
    // Read the five variable values
    espSerial.readBytes((uint8_t*)&temperature, sizeof(temperature));
    delay(10);
    espSerial.readBytes((uint8_t*)&humidity, sizeof(humidity));
    delay(10);
    espSerial.readBytes((uint8_t*)&airQuality, sizeof(airQuality));
    delay(10);
    espSerial.readBytes((uint8_t*)&voltage, sizeof(voltage));
    delay(10);
    espSerial.readBytes((uint8_t*)&current, sizeof(current));
    delay(10);
    // Wait until end of frame marker is received
    while (espSerial.read() != '*');

    // Process received data
    Serial.print("Temperature: ");
    Serial.println(temperature);
    Serial.print("Humidity: ");
    Serial.println(humidity);
    Serial.print("Air Quality: ");
    Serial.println(airQuality);
    Serial.print("AC Voltage: ");
    Serial.println(voltage);
    Serial.print("Current: ");
    Serial.println(current);
    Serial.println();

    sendDataToDatabase(temperature, humidity, voltage, current,
airQuality);
  }

  delay(15000); // Interval
}

void sendDataToDatabase(float temperature, float humidity, float voltage,
float current, float airQuality) {
  if (netClient.connect(serverAddress, 80)) {
    Serial.println("Connected to server");
    // Make an HTTP request
    netClient.print(String("GET
http://xxx.xxx.xxx.xxx/iot_project/sql_query.php?") +
                    "temperature=" + String(temperature) +
```

```
                    "&humidity=" + String(humidity) +
                    "&acVolts=" + String(voltage) +
                    "&current=" + String(current) +
                    "&airQuality=" + String(airQuality) +
                    " HTTP/1.1\r\n" +
                    "Host: " + serverAddress + "\r\n" +
                    "Connection: close\r\n\r\n");

    unsigned long timeout = millis();
    while (netClient.available() == 0) {
      if (millis() - timeout > 1000) {
        Serial.println(">>> Client Timeout !");
        netClient.stop();
        return;
      }
    }

    // Read all the lines of the reply from server and print them to
Serial
    while (netClient.available()) {
      String line = netClient.readStringUntil('\r');
      Serial.print(line);
    }
  } else {
    Serial.println("Connection failed");
  }
}
```

- **Main Receiver Node**

```
#include <esp_now.h>
#include <WiFi.h>

// Structure definition to receive data
typedef struct {
  int id;
  float x;
  float y;
} SensorData;

SensorData receivedData;

// Create structures to hold readings from each board
SensorData board1Data;
```

```
SensorData board2Data;
SensorData board3Data;

// Array of all structures
SensorData boards[3] = {board1Data, board2Data, board3Data};

// SMS parameters
String smsStart = "AT+CMGS=\"+92";
String smsEnd = "\"\r";
String userPhoneNumber = "xxxxxxxxxx";

float temperature;
float humidity;
float airQuality;
float voltage;
float current;

const int tempThreshold = 35;
const int humidThreshold = 90;
const int airQualityThreshold = 2500;
const int voltageThreshold = 270;
const float currentThreshold = 1.5;

// Callback function to be executed when data is received
void onDataReceive(const uint8_t *macAddr, const uint8_t *incomingData,
int len) {
  char macStr[18];
  snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x",
           macAddr[0], macAddr[1], macAddr[2], macAddr[3], macAddr[4],
macAddr[5]);

  memcpy(&receivedData, incomingData, sizeof(receivedData));

  int id = receivedData.id;
  Serial.print("ID is: "); Serial.println(id);

  switch(id) {
    case 1:
      boards[id - 1].x = receivedData.x;
      boards[id - 1].y = receivedData.y;
      temperature = boards[id - 1].x;
      humidity = boards[id - 1].y;
      if ((temperature >= 0) && (temperature <= 80)) {
        Serial.print("Temperature is: "); Serial.println(temperature);
      }
```

```
      if ((humidity >= 0) && (humidity <= 100)) {
        Serial.print("Humidity is: "); Serial.println(humidity);
      }
      break;

    case 2:
      boards[id - 1].x = receivedData.x;
      airQuality = boards[id - 1].x;
      if ((airQuality >= 0) && (airQuality <= 5000)) {
        Serial.print("Air Quality reading is: ");
Serial.println(airQuality);
      }
      break;

    case 3:
      boards[id - 1].x = receivedData.x;
      boards[id - 1].y = receivedData.y;
      voltage = boards[id - 1].x;
      current = boards[id - 1].y;
      if ((voltage >= 0) && (voltage <= 300)) {
        Serial.print("AC Voltage is: "); Serial.println(voltage);
      }
      if ((current >= 0) && (current <= 500)) {
        Serial.print("AC Current is: "); Serial.println(current);
      }
      break;
  }

  Serial.println();

  // Send all variable values to Arduino Uno via SoftwareSerial
  Serial.write('#');  // Start of frame marker
  delay(10);
  Serial.write((uint8_t*)&temperature, sizeof(temperature));
  delay(10);
  Serial.write((uint8_t*)&humidity, sizeof(humidity));
  delay(10);
  Serial.write((uint8_t*)&airQuality, sizeof(airQuality));
  delay(10);
  Serial.write((uint8_t*)&voltage, sizeof(voltage));
  delay(10);
  Serial.write((uint8_t*)&current, sizeof(current));
  delay(10);
  Serial.write('*');  // End of frame marker
```

```
  if (temperature > tempThreshold) {
    sendSMS("Temperature", temperature);
  }
  if (humidity > humidThreshold) {
    sendSMS("Humidity", humidity);
  }
  if (airQuality > airQualityThreshold) {
    sendSMS("Air Quality", airQuality);
  }
  if (voltage > voltageThreshold) {
    sendSMS("Voltage", voltage);
  }
  if (current > currentThreshold) {
    sendSMS("Current", current);
  }
}

void setup() {
  Serial.begin(9600);
  Serial2.begin(9600);

  // Set device as a Wi-Fi Station
  WiFi.mode(WIFI_STA);
  Serial.print("ESP Board MAC Address: ");
  Serial.println(WiFi.macAddress());

  // Initialize ESP-NOW
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }

  // Register receive callback function
  esp_now_register_recv_cb(onDataReceive);
  delay(3000);
}

void loop() {
  delay(1000);
}

void sendSMS(String parameter, float value) {
  Serial2.println("AT+CMGF=1");
  delay(500);
  Serial2.println(smsStart + userPhoneNumber + smsEnd);
```

```
  delay(2500);
  Serial2.println("ALERT");
  Serial2.print(parameter + " is: "); Serial2.println(value);
  delay(500);
  Serial2.println((char)26);  // ASCII code for CTRL+Z
  delay(500);
}
```

- **Temperature/Humidity Slave Node**

```
#include <esp_now.h>
#include <WiFi.h>
#include "DHT.h"

#define DHT_PIN 23   // Digital pin connected to the DHT sensor
#define DHT_TYPE DHT11   // DHT 11 sensor type

// MAC Address of the receiver
uint8_t receiverMACAddress[] = {0x08, 0xD1, 0xF9, 0x27, 0xF3, 0x88};

// Define a structure for the data to be sent
typedef struct {
  int id;   // Unique identifier for each sender board
  float temperature;
  float humidity;
} SensorData;

SensorData sensorData;

esp_now_peer_info_t peerInfo;
DHT dht(DHT_PIN, DHT_TYPE);

// Callback function when data is sent
void onDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  Serial.print("\r\nLast Packet Send Status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" :
"Delivery Fail");
}

void setup() {
  // Initialize Serial Monitor
  Serial.begin(115200);

  // Set device as a Wi-Fi Station
  WiFi.mode(WIFI_STA);

  // Initialize ESP-NOW
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }
```

```
  // Register callback function for when data is sent
  esp_now_register_send_cb(onDataSent);

  // Register peer information
  memcpy(peerInfo.peer_addr, receiverMACAddress, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  // Add peer
  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Failed to add peer");
    return;
  }

  // Initialize the DHT sensor
  dht.begin();
}

void loop() {
  // Reading temperature and humidity takes about 250 milliseconds
  float humidity = dht.readHumidity();
  float temperatureC = dht.readTemperature();
  float temperatureF = dht.readTemperature(true);

  // Check if any reads failed and exit early to try again
  if (isnan(humidity) || isnan(temperatureC) || isnan(temperatureF)) {
    Serial.println(F("Failed to read from DHT sensor!"));
    return;
  }

  // Compute heat index in Fahrenheit and Celsius
  float heatIndexF = dht.computeHeatIndex(temperatureF, humidity);
  float heatIndexC = dht.computeHeatIndex(temperatureC, humidity, false);

  // Print sensor readings to Serial Monitor
  Serial.print(F("Humidity: "));
  Serial.print(humidity);
  Serial.print(F("%  Temperature: "));
  Serial.print(temperatureC);
  Serial.print(F("°C "));
  Serial.print(temperatureF);
  Serial.print(F("°F  Heat index: "));
  Serial.print(heatIndexC);
  Serial.print(F("°C "));
  Serial.print(heatIndexF);
  Serial.println(F("°F"));

  // Set data to be sent
  sensorData.id = 1;
  sensorData.temperature = temperatureC;
  sensorData.humidity = humidity;

  // Send data via ESP-NOW
```

```
  esp_err_t result = esp_now_send(receiverMACAddress, (uint8_t *)
&sensorData, sizeof(sensorData));

  if (result == ESP_OK) {
    Serial.println("Sent successfully");
  } else {
    Serial.println("Error sending data");
  }

  delay(5000);  // Wait for 5 seconds before the next reading
}
```

- **Air Quality Slave Node**

```
#include <esp_now.h>
#include <WiFi.h>

#define MQ135_PIN 36  // Define the pin connected to the MQ135 sensor

// Replace with the receiver's MAC Address
uint8_t receiverMACAddress[] = {0x08, 0xD1, 0xF9, 0x27, 0xF3, 0x88};

// Structure to send data
typedef struct {
  int id;  // Unique identifier for each sender board
  float airQuality;
  float randomValue;
} SensorData;

SensorData sensorData;
esp_now_peer_info_t peerInfo;

float defaultSensorValue = random(1, 10);  // Initial random value for
sensor
float airQuality;
float sensorReading;

// Callback when data is sent
void onDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  Serial.print("\r\nLast Packet Send Status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" :
"Delivery Fail");
}

void setup() {
  // Initialize Serial Monitor
  Serial.begin(115200);
```

```
  // Set device as a Wi-Fi Station
  WiFi.mode(WIFI_STA);

  // Initialize ESP-NOW
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }

  pinMode(MQ135_PIN, INPUT);

  // Register callback function for when data is sent
  esp_now_register_send_cb(onDataSent);

  // Register peer information
  memcpy(peerInfo.peer_addr, receiverMACAddress, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  // Add peer
  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Failed to add peer");
    return;
  }
}

void loop() {
  sensorReading = analogRead(MQ135_PIN);

  // Use default value if sensor reading is zero
  if (sensorReading == 0) {
    sensorReading = defaultSensorValue;
  } else {
    defaultSensorValue = sensorReading;
  }

  airQuality = sensorReading;
  Serial.print("Air Quality: ");
  Serial.println(airQuality);

  // Set values to send
  sensorData.id = 2;
  sensorData.airQuality = airQuality;
  sensorData.randomValue = random(0, 4);
```

```
  // Send message via ESP-NOW
  esp_err_t result = esp_now_send(receiverMACAddress, (uint8_t *)
&sensorData, sizeof(sensorData));
  if (result == ESP_OK) {
    Serial.println("Sent successfully");
  } else {
    Serial.println("Error sending the data");
  }

  delay(5000);  // Wait for 5 seconds before the next reading
}
```

- **Voltage/Current Slave Node**

```
#include <esp_now.h>
#include <WiFi.h>

// Replace with the receiver's MAC address
uint8_t receiverMACAddress[] = {0x08, 0xD1, 0xF9, 0x27, 0xF3, 0x88};

// Structure to send data
typedef struct {
  int id; // Unique identifier for each sender board
  float voltage;
  float current;
} SensorData;

SensorData sensorData;
esp_now_peer_info_t peerInfo;

#define VOLTAGE_PIN 39
#define CURRENT_PIN 35

float AC_voltage = 0;
float sensorReading = 0;
float averageVoltage = 0;
float voltageReading = 0;

float peakVoltage;
float peakCurrentThroughResistor;
float RMSCurrentThroughResistor;
float RMSCurrentInWire;

// Callback when data is sent
void onDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
```

```
  Serial.print("\r\nLast Packet Send Status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" :
"Delivery Fail");
}

void setup() {
  // Initialize Serial Monitor
  Serial.begin(115200);

  // Set device as a Wi-Fi Station
  WiFi.mode(WIFI_STA);

  // Initialize ESP-NOW
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }

  // Register callback function for when data is sent
  esp_now_register_send_cb(onDataSent);

  // Register peer information
  memcpy(peerInfo.peer_addr, receiverMACAddress, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  // Add peer
  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Failed to add peer");
    return;
  }

  pinMode(VOLTAGE_PIN, INPUT);
  pinMode(CURRENT_PIN, INPUT);
}

void loop() {
  for (int i = 0; i < 30; i++) {
    sensorReading = analogRead(VOLTAGE_PIN);
    voltageReading = (sensorReading / 4096.0) * 300; // Convert the analog
reading to voltage (0 - 300V)
    averageVoltage += voltageReading;
  }

  AC_voltage = averageVoltage / 30;  // Calculate average voltage
```

```
  averageVoltage = 0;

  // Calculate current
  peakVoltage = getPeakVoltage();

  // Use Ohm's law to calculate current through the resistor in mA
  peakCurrentThroughResistor = (peakVoltage / 110.0) * 1000.0;

  // Convert peak current to RMS current for sine wave
  RMSCurrentThroughResistor = peakCurrentThroughResistor * 0.707;

  Serial.print("AC Voltage: ");
  Serial.println(AC_voltage);
  Serial.print("AC Current: ");
  Serial.println(RMSCurrentThroughResistor);

  // Set values to send
  sensorData.id = 3;
  sensorData.voltage = AC_voltage;
  sensorData.current = RMSCurrentThroughResistor * 100; // Multiply by 100
to avoid float issues during transmission

  // Send message via ESP-NOW
  esp_err_t result = esp_now_send(receiverMACAddress, (uint8_t *)
&sensorData, sizeof(sensorData));

  if (result == ESP_OK) {
    Serial.println("Sent with success");
  } else {
    Serial.println("Error sending the data");
  }

  delay(5000); // Wait for 5 seconds before next reading
}

float getPeakVoltage() {
  float result;
  int readValue; // Value read from the sensor
  int maxValue = 0; // Store max value here
  uint32_t startTime = millis();

  // Sample for 3 seconds
  while ((millis() - startTime) < 3000) {
    readValue = analogRead(CURRENT_PIN);
    // Check for a new max value
```

```
    if (readValue > maxValue) {
      maxValue = readValue;
    }
  }

  // Convert the max digital value to voltage
  result = (maxValue * 3.3) / 4096.0; // Convert max value to voltage (0 -
3.3V)
  return result;
}
```

Anx-B

Software Code

- **App.js**

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { BrowserRouter as Router, Route, Link } from 'react-router-dom';

const TemperatureContainer = () => {
  const [temperature, setTemperature] = useState(0);

  useEffect(() => {
    const fetchTemperature = async () => {
      try {
        const response = await axios.get('/api/temperature');
        setTemperature(response.data.temperature);
      } catch (error) {
        console.error('Error fetching temperature data:', error);
      }
    };
    fetchTemperature();
  }, []);

  return (
    <div className="temperature-container">
      <h2>Temperature</h2>
      <p>{temperature} °C</p>
    </div>
  );
};

const HumidityContainer = () => {
  const [humidity, setHumidity] = useState(0);
```

```
  useEffect(() => {
    const fetchHumidity = async () => {
      try {
        const response = await axios.get('/api/humidity');
        setHumidity(response.data.humidity);
      } catch (error) {
        console.error('Error fetching humidity data:', error);
      }
    };
    fetchHumidity();
  }, []);

  return (
    <div className="humidity-container">
      <h2>Humidity</h2>
      <p>{humidity} %</p>
    </div>
  );
};

const VoltageContainer = () => {
  const [voltage, setVoltage] = useState(0);

  useEffect(() => {
    const fetchVoltage = async () => {
      try {
        const response = await axios.get('/api/voltage');
        setVoltage(response.data.voltage);
      } catch (error) {
        console.error('Error fetching voltage data:', error);
      }
    };
    fetchVoltage();
  }, []);

  return (
    <div className="voltage-container">
      <h2>Voltage</h2>
      <p>{voltage} V</p>
    </div>
  );
};

const CurrentContainer = () => {
  const [current, setCurrent] = useState(0);

  useEffect(() => {
    const fetchCurrent = async () => {
      try {
        const response = await axios.get('/api/current');
        setCurrent(response.data.current);
      } catch (error) {
        console.error('Error fetching current data:', error);
      }
```

```
    };
    fetchCurrent();
  }, []);

  return (
    <div className="current-container">
      <h2>Current</h2>
      <p>{current} A</p>
    </div>
  );
};

const AirQualityContainer = () => {
  const [airQuality, setAirQuality] = useState(0);

  useEffect(() => {
    const fetchAirQuality = async () => {
      try {
        const response = await axios.get('/api/airquality');
        setAirQuality(response.data.airQuality);
      } catch (error) {
        console.error('Error fetching air quality data:', error);
      }
    };
    fetchAirQuality();
  }, []);

  return (
    <div className="airquality-container">
      <h2>Air Quality</h2>
      <p>{airQuality}</p>
    </div>
  );
};

const Logs = () => {
  const [logs, setLogs] = useState([]);
  const [currentPage, setCurrentPage] = useState(1);
  const rowsPerPage = 10;

  useEffect(() => {
    const fetchLogs = async () => {
      try {
        const response = await axios.get('/api/logs');
        setLogs(response.data);
      } catch (error) {
        console.error('Error fetching logs:', error);
      }
    };
    fetchLogs();
  }, []);

  const handlePageChange = (pageNumber) => {
    setCurrentPage(pageNumber);
```

```
  };

  const currentLogs = logs.slice((currentPage - 1) * rowsPerPage,
currentPage * rowsPerPage);

  return (
    <div className="logs-container">
      <h2>Logs</h2>
      <table>
        <thead>
          <tr>
            <th>Timestamp</th>
            <th>Temperature</th>
            <th>Humidity</th>
            <th>Current</th>
            <th>Air Quality</th>
            <th>AC Volts</th>
          </tr>
        </thead>
        <tbody>
          {currentLogs.map((log, index) => (
            <tr key={index}>
              <td>{log.Timestamp}</td>
              <td>{log.temperature}</td>
              <td>{log.humidity}</td>
              <td>{log.current}</td>
              <td>{log.airQuality}</td>
              <td>{log.acVolts}</td>
            </tr>
          ))}
        </tbody>
      </table>
      <div className="pagination">
        {Array.from({ length: Math.ceil(logs.length / rowsPerPage) }, (_,
index) => (
          <button key={index} onClick={() => handlePageChange(index + 1)}>
            {index + 1}
          </button>
        ))}
      </div>
    </div>
  );
};

const App = () => (
  <Router>
    <div>
      <nav>
        <ul>
          <li>
            <Link to="/">Dashboard</Link>
          </li>
          <li>
            <Link to="/logs">Logs</Link>
```

```
          </li>
        </ul>
      </nav>
      <Route path="/" exact component={TemperatureContainer} />
      <Route path="/humidity" component={HumidityContainer} />
      <Route path="/voltage" component={VoltageContainer} />
      <Route path="/current" component={CurrentContainer} />
      <Route path="/airquality" component={AirQualityContainer} />
      <Route path="/logs" component={Logs} />
    </div>
  </Router>
);

export default App;
```