Wireless Sensor Network Design for Home Automation

A final report presented to the Technological University Dublin – Tallaght Campus

by

Carl Surban Sagrado

X00084403

Bachelor of Engineering in Electronic Engineering (Honours)

Semester 7

4th May 2021

Department of Electronic Engineering

Supervisor: Mr. Michael Gill

Wireless Sensor Network Design for Home Automation

Submitted by Carl Sagrado, X00084403

Preface

The final report was made during the last semester of the fourth year as part of the final year project, a requirement for the completion of the Electronic Engineering (Hons) Bachelor of Engineering study at Technological University Dublin – Tallaght Campus, Dublin, Ireland.

Abstract

This report presents the development of a wireless sensor network (WSN) prototype system for indoor monitoring and automation of a residential building. The system will autonomously collect environmental conditions from remote units and control these units. The system will read and publish sensor data in real-time on a web interface. The main purpose of the system is to monitor environmental conditions to aid in reducing energy consumption. The platform also provides an interface to control remote devices, i.e., relays, which will be controlled wirelessly using the same web interface.

Keywords: **Wireless Sensor Networks**, **Home Automation**, **Web Interface**, **Smart Devices, IoT**

Table of Contents

Acknowledgements

First, I wish to thank my project supervisor Mr. Michael Gill for his unlimited guidance, information, and materials throughout the development of this project. His support, insight and guidance has brought enormous influence for the completion of the project prototype. I would also like to extend my thanks to Mr. Thomas Murray for me providing different ideas and additional support. Thanks especially to Mr. Hugh Brennan and Mr. Bill Black for providing unconditional financial support throughout my academic journey. Finally, thanks to my friends and family for their support throughout.

Declaration

I hereby certify that the material, which I now submit is entirely my own work and has not been taken from the work of others except to the extent that such work has been cited and acknowledged within the text of my own work.

Signature: 

Date: 04/05/2021

List of Figures

List of Tables

NOMENCLATURE

The list below is some of abbreviations used in this report.

ACK	:	Acknowledgement
ADC	:	Analog Digital Converter
CO2	:	Carbon Dioxide
CSS	:	Cascading Style Sheet
GPIO	:	General Purpose Input Output
GUI	:	Graphic User Interface
HTML	:	Hyper Text Markup Language
HTTP	:	Hypertext Transfer Protocol
IoT	:	Internet of Things
IP	:	Internet Protocol
JS	:	JavaScript
MAC	:	Media Access Control
NFC	:	Near-Field-Communication
PCB	:	Printed Circuit Board
SoC	:	Software on Chip
TCP	:	Transmission Control Protocol
Wi-Fi	:	Wireless Fidelity
WSN	:	Wireless Sensor Network

Chapter 1 – Introduction

Wireless Sensor Networks (WSN) is commonly defined as a collection of small powered wireless nodes equipped with sensors. These sensor nodes are used to actively monitor physical phenomena such as temperature, humidity, CO2, vibration, etc. and produce sensory data. The nodes are grouped together in a wireless network and facilitates collection information of its physical environment. Typical communication structure of the nodes involves forwarding of data through other nodes to a gateway or a base station which forwards data to a computer device on the network.

Using wireless technology can allow greater complexity over wired transmission and helps facilitate the installation of sensors, controllers, and actuators. At present, home automation and monitoring are the dominant applications of wireless sensor networks where the idea of automating the living space using wireless systems and sensors has become a commonplace.

This report presents the development of a wireless sensor network (WSN) prototype system for indoor monitoring and automation of a residential building. The developed sensor network will be used to detect the environmental conditions of a residential unit and enable control of wireless devices, remotely.

The first section of the report highlights the overview of the project and discusses different parts of the project that needs development, as well as the key objectives and timing delivery of the proposed project. The second section examines the technologies used to develop the prototype. The section examines the background of the relevant hardware involved to create the wireless sensor network as well as different software technologies used to develop web application for the project.

Section three and four presents detailed documentation of the design and implementation of the project containing details of the implemented software and hardware technologies with explanations of the method used in implementing the design of the project. The remaining section of this report presents future ideas that may be used to improve the prototype, engineering issues concerning electronic assembly and the concluding discussion regarding the project.

1.1 Project Overview

Proposed project activity is to develop the necessary components to enable a fully functional wireless sensor network to contribute to home automation. The entire WSNs project typically and importantly comprises of 3 sections:

1. **Sensor Networks** comprises of small powered intelligent devices with wireless infrastructure that collects and transfers sensory data to a web server for use and analysis. The same network device technology is used to adjust and control devices which are remotely executed by the user via web interface.
2. **Web Server** is responsible for receiving and publishing of data from the sensor networks to a web interface and transmitting remote data from the web interface to the remote device of the network.
3. **Web Interface** is a system web page that displays all sensory data retrieved from the sensor networks and hosts an interface to control remote devices.

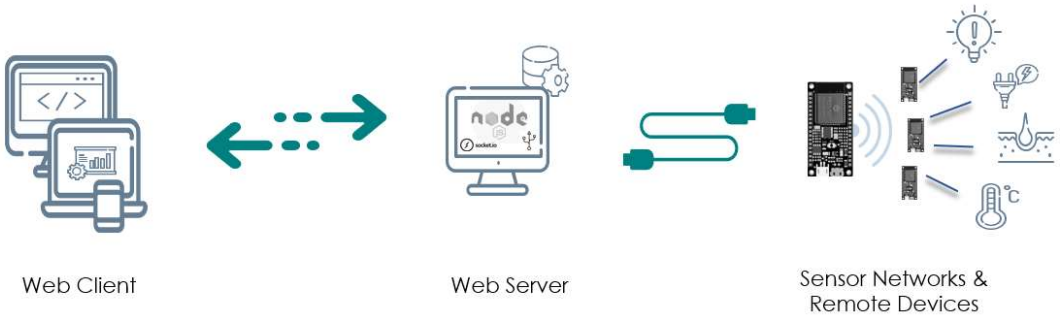

Figure 1: Diagram of the Wireless Sensor Networks Project

The diagram shown in figure 1 is the overview of the concept design for the entire Wireless Sensor Networks Project. Each section shown in the diagram will be developed in parallel with each other to accommodate the development of different features set for this project. For example, the Web Server is expected to store, evaluate, and communicate sensor data with remote commands between a web client over the sensor network. Sensor nodes are expected to deliver real-time sensor readings to the master node and perform instructions set by the user, etc.

As each section will be developed separately, the final activity of the project is to test the complete operation of the project by deploying the sensor nodes in 3 different rooms allocated with a different set of sensors and/or remote relays and observing the sensor data via a web interface.

1.3 Project Planning

Details shown in Table 1 overleaf, expands the plan laid out for the semester. The design and implementation of the project will take the full semester to complete. A prototype product is expected to function accordingly which will be mentioned in this report. For each set of weeks, there are tasks and activities which are expected to be completed. These should enable learning and familiarization of the different technologies which will be required to complete the project prototype.

Table 1: Project Plan

<table>
<tr><th>Week</th><th>Date</th><th>Description</th></tr>
<tr><td>1</td><td>25/01/21</td><td>Description of the Project</td></tr>
<tr><td>2</td><td>01/02/21</td><td rowspan="5">Web Interface Design
Introduction to -:
• HTML
• CSS (Cascading Style Sheets)
• DOM (Document Object Model)
• JavaScript
• jQuery

Coding practice by developing a Calculator leading to the design of a web user interface.</td></tr>
<tr><td>3</td><td>08/02/21</td></tr>
<tr><td>4</td><td>15/02/21</td></tr>
<tr><td>5</td><td>22/02/21</td></tr>
<tr><td>6</td><td>01/03/21</td></tr>
<tr><td>7</td><td>08/03/21</td><td rowspan="3">Web Server
Introduction to -:
• NodeJS
• Socket.io
• Serialport.io
Development of the Web Server to handle client request and sensor data received via serial connection.</td></tr>
<tr><td>8</td><td>15/03/21</td></tr>
<tr><td>9</td><td>22/03/21</td></tr>
<tr><td>10</td><td>29/03/21</td><td rowspan="3">Sensor Networks
Introduction to -:
• NodeJS
• Socket.io
• Serialport.io

*Development of the 4 sensor nodes using ESP32.
*Rapid prototyping and testing of the project.</td></tr>
<tr><td>11</td><td>04/04/21</td></tr>
<tr><td>12</td><td>11/04/21</td></tr>
</table>

1.4 Methodology

1.4.1 Design Methodology

The design is based on the specific requirements as outlined in the overview of this project. Each section of the project is to be developed individually and expected to work independently to enable different means of testing. To develop a functional prototype system, each section will be integrated progressively to other section of the project leading up to a complete system. The greater advantage of an application-specific approach is designing a system to solve one task and not designing a generic system which can ultimately impede the entire progress of the project. By taking this approach the design becomes more focused on solving the task at hand. The design should also allow for software and hardware modifications to any aspect of the system. This ensures that the system can be modified, changed and calibrated to meet the desired outcome during rapid development testing and maintains flexibility in the system configuration and implementation. The sensor networks ought to be designed as a semi-automated system with minimal input from the user after the initial setup completion and interactable via web application, incorporated with measures to detect failures and errors during operation.

1.4.2 Project Development

The initial design and development of the Wireless Sensor Network System will mostly be initiated and tested individually. This is to ensure that each node works as intended allowing prompt investigation and modification to meet expected result before the prototype is rolled out. Any significant process and technologies used to develop this project will be discussed in the design section of this report accompanied with relevant information regarding their relevance.

Chapter 2 – Background

This section discusses available information and technologies used pertaining to this project.

2.1 Sensor Networks

Sensor networks are a collection of small powered intelligent devices equipped with wireless infrastructure and sensors that perform the collection of environmental or physical sensory data and transmission of relative data to a computer network for use and analysis. See figure 2 for the senor node structure.

Sensor Node

Typical structure of a sensor node consists of the following:

- Microcontroller
- Communication module
- Sensors
- Power source

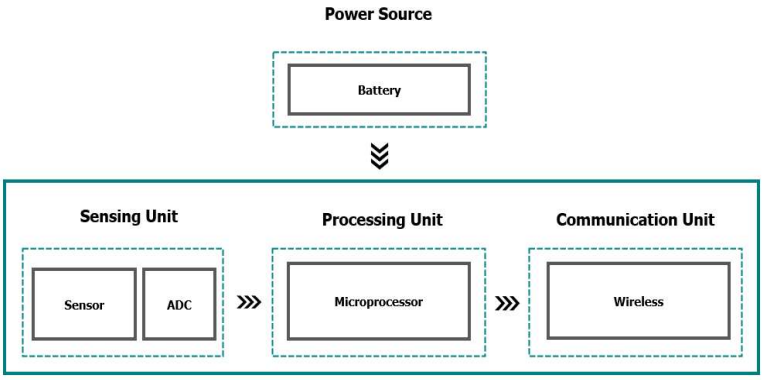

Figure 2: Sensor Node Structure

Microcontroller –

The microcontroller is a self-contained programmable device equipped with a processor, memory, and other peripherals. It is an IC chip that contains embedded instruction to execute an instruction to control and enable specific applications.

Most manufacturers of microcontrollers supply a wide range of ultra-low powered 8-bit, 16-bit and 32-bit devices which comes in a varying form factor. The different microcontroller features different sets of peripherals made suitable for various applications. The microcontroller chosen to be used in this project is a type of that requires low supply-voltage that is capable of outputting mixed signals with wireless communication capabilities known as '**ESP-32'** shown in figure 3 and relevant information of the microcontroller will be discussed throughout this report. This microcontroller will serve as the brain for each sensor node programmed with a set of instructions according to its specific purpose and role in the project.

Figure 3: ESP32 SoC Physical Chip

Wireless Communication module -

For a sensor node to communicate wirelessly, requires a module or a system to enable such communication. There are several available wireless systems which have been adopted for

various wireless sensor network projects, such technologies include, but not limited to, Bluetooth, NFC, Wi-Fi, etc. The pre-made ESP-32 SoC module assembly comes with a pre-tuned PCB antenna required to enable wireless connectivity such as Bluetooth and Wi-Fi. Shown below in figure 4, depicts the functional block diagram of the ESP32 chip module with its integrated components and features.

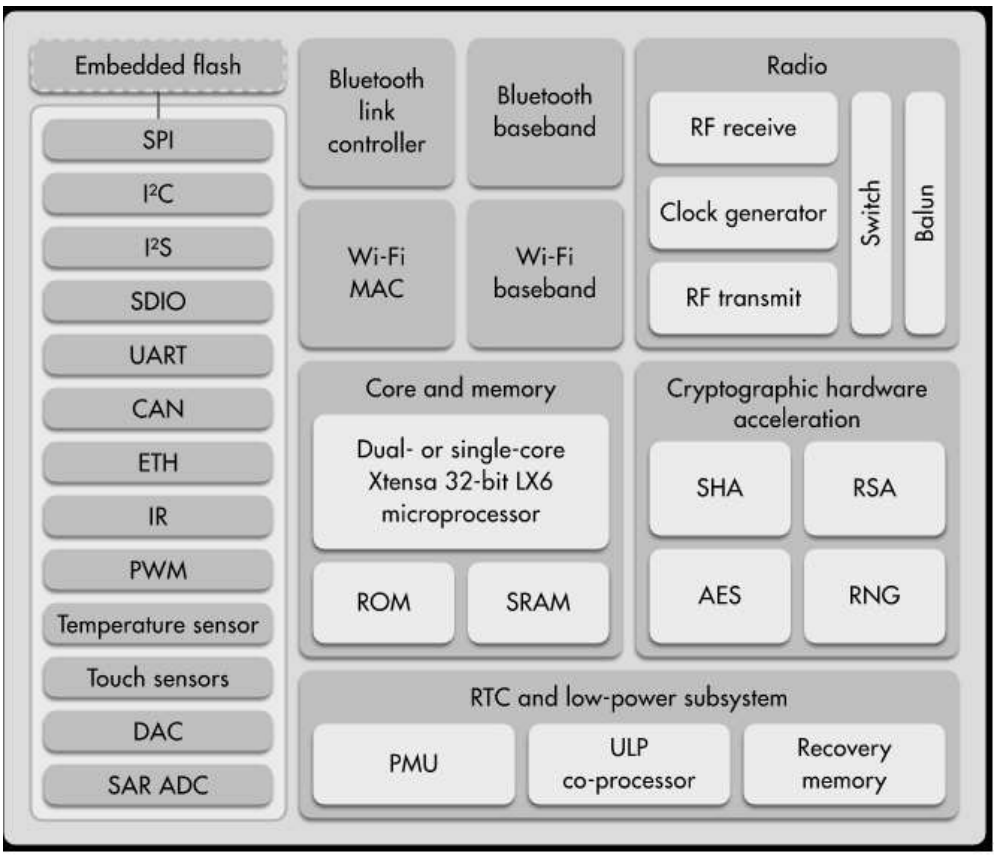

Figure 4: Functional block diagram of the ESP32 from the Datasheet [1]

Wi-Fi – ESP32 SoC module chip uses a TCP/IP and a full 802.11 b/g/n Wi-Fi MAC Protocol which supports the Basic Service Set (BSS), a component of the IEE 802.11 WLAN architecture, enabling STA and SoftAP mode operations. The module offers Wi-Fi baseband and Wi-Fi MAC functionalities and capable of running both functionalities simultaneously.

Bluetooth – The module comes with Bluetooth link controller and Bluetooth baseband which deliver via baseband protocols and low-level link routines relevant for different applications. The Bluetooth interface provides UART HCI interface, supporting up to 4Mps and provides SDIO/SPI HCI Interface. Both Bluetooth technologies support multiple connections and other operations such as inquiry, page and secure simple pairing etc. provides power management low-power applications.

ESP-32 is a low powered system on a chip microcontroller with a single 2.4GHz W-Fi and Bluetooth combo designed to provide the best power performance, robust and reliable performance suitable for a wide range of IoT applications.

For prototyping this project, ESP-32 general-purpose development board will be used which packs with all the necessary features required for the sensor nodes to operate including, computing power, wireless capabilities, I/O connections, and in-built low power management incorporated into one single development board.

ESP32 offers 36 GPIO pins, some of which are configurable for different types of applications, see Figure 5.

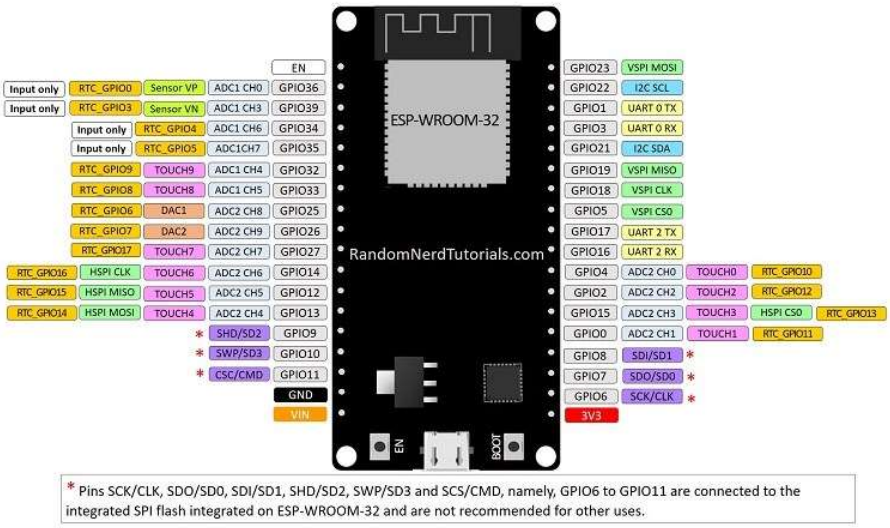

Figure 5: ESP32 Development Board Pinouts [2]

The ESP-32 chip itself operates at 3.3V, and the development board comes with a power supply regulator circuit ensuring a supply of DC power to the main chip is regulated. For prototyping development purposes, the board is powered via micro-USB connection and allow the in-built regulator circuit to regulate the supply from 5V to 3.3V to power the chip. The physical layout of the module is shown in figure 6 below.

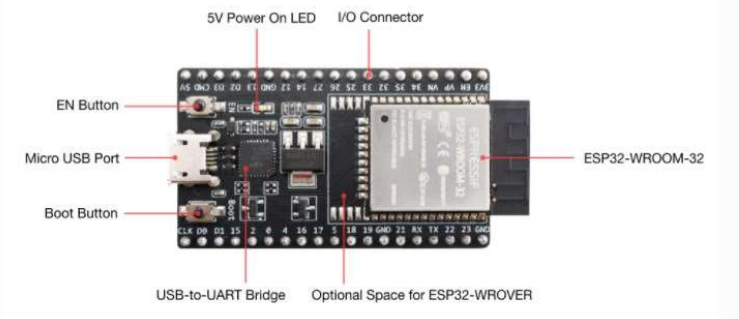

Figure 6: ESP32 Development Kit Physical Layout [13]

Sensors -

Sensors are transducer devices that convert energy from one domain to another. Such devices convert the quantity to be sensed into a measurable signal. Since the signal conditioning and processing is being carried out by electronic circuits, the relevant output of transducer devices is generally voltages or currents.

DHT11 Module– shown in Figure 7, is a temperature and humidity sensor module. The module features a dual-sensing capability to a digital signal output, delivered from a resistive-type humidity measurement component and an NTC temperature measurement component. The module is a low powered and cost-effective device capable of delivering the required operation for temperature and humidity sensing for our project prototype.

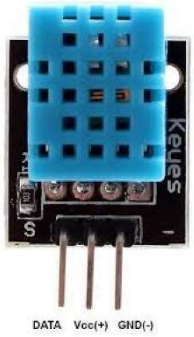

Figure 7: DHT11 Pinouts [14]

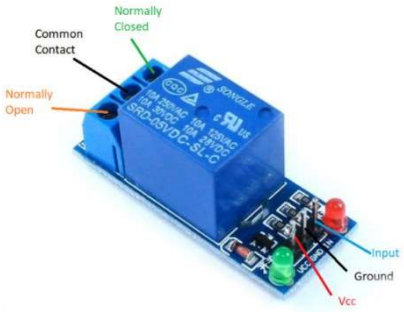

Figure 8: Relay Module Pinouts [15]

RELAY Module – Relay switch is an electrically operated switch which is controlled by low-power signal primarily used for controlling high powered applications. The incorporation of relay modules to the project enables the control of the end device, remotely.

For remote switching capabilities, a 5V one channel relay module is used for the prototype. The device requires 5V to operate and capable to handle up a load up to 250V/10A AC and 30V/10A DC. Pinout of the module is shown in fig 8.

SCD30 Sensor Module – is an NDIR CO2 sensor module capable of detecting CO2 concentrations up to 40,000 parts per million [3] with great accuracy when operating at room temperature. The module is operational at 3.3V and draws current, of 19mA on average.

Carbon dioxide is a key indicator of indoor air quality. While it presents as a natural component of fresh air, higher concentration of CO2 exposure can lead to serious health issues. Typical CO2 concentrations of occupied indoor spaces range between 400-1000 ppm. Concentrations over 1000 ppm in an indoor environment is a potential indicator of poor air exchange. Constant exposure of high concentrations CO2 can produce a variety of health effects which may include, headaches, dizziness, restlessness, difficulty breathing sweating, tiredness, etc.

Software Libraries

The integrated development environment (IDE) used to program and communicate to the ESP32 board is called Arduino IDE. This IDE is an open-source cross-platform primarily released for Arduino project has been made compatible with ESP board through additional package installation of EspressIf IoT Development Framework, by EspressIf.

ESP-NOW – is a wireless protocol designed by EspresIf which uses a 2.4GHz frequency for wireless connectivity, similar to the standard Wi-Fi. Unlike the standard Wi-Fi that uses IP addresses assigned to the devices to communicate, The ESP-NOW uses the MAC address of the sender and the receiver to communicate.

The ESP-NOW library available for ESP32 and ESP8266 devices was developed by EspressIf providing a connectionless communication protocol, an ideal solution such as our application. The library features a peer-to-peer short packet transmission where a linked list of device and peer device information is maintained at the low-level layer, which is used to send and receive data. ESP-NOW doesn't require handshake as it peer-to-peer connectivity and applies some of the IEEE802.11 standards for wireless communication.

2.2 Web Server

A web server is a software application that typically handles web requests by web clients and returns web pages in response to requests. For the prototype development, a web server serves for two particular purposes, data handling from the sensors network and handling web client requests. There are different technologies required to enable a fully functional server, which usually falls into two categories, namely:

- Backend – handles the processing of data processing, exchange and storing.
- Frontend – handles the presentation of data, navigation, and styling.

A typical example of an application of a web server is depicted in *figure 9*.

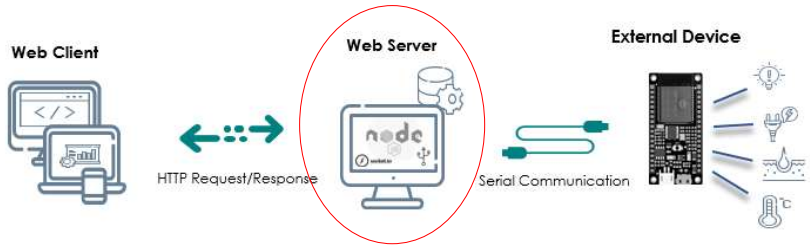

Figure 9: Web server application

External device could be anything that provides data to a web server via any means of communication and the server processes data supplied and presents accordingly to web clients using the HTTP protocol.

HyperText Transfer Protocol (HTTP) – is an application-layer protocol designed for communication between web browsers and web servers. This protocol allows fetching of resources, typically HTML documents through an HTTP request/response transaction. The HTTP request is initiated by a web browser to a webserver and HTTP response is the response sent by the server according to the request made by the web browser.

NodeJS (Node) is an open-source cross-platform runtime environment capable of delivering the backend services for different types of application. Node is ideal for high-scalable, data-intensive and real-time backend services that power client applications. Node application uses *JavaScript* as a programming language for the server-side development with a large ecosystem of open-source libraries.

For a server to successfully host services, such as data exchange handling to and from an external source, request handling, presenting data, HTTP request handling, etc. requires the application of different modules and frameworks such as express, socket.io, serial port, etc. (Further details about different the modules mentioned in the development section of this report).

2.3 Web Interface

A web interface is an application that allows users to interact with the contents of an application running on a web server through a web browser. Modern web interfaces commonly developed with the following: *HTML, CSS and JavaScript*.

HyperText Markup Language (HTML) - is a standard markup language for creating web pages. A standardized system that describes the structure of a web page with a series of elements containing instructions for web browsers on how to display contents.

Cascading Style Sheets (CSS) – is a styling language used for an HTML document. It describes how elements in HTML are presented and defines the formation of layouts to which improves content accessibility and presentation flexibility.

JavaScript (JS) – a unique programming language that is applicable to run on a web browser and any other applications for web development. JavaScript transforms static web pages to dynamic web pages by manipulating contents through HTML's Document Object Model (DOM), a programmatic data on a web page.

Chapter 3 – Design

This chapter discusses the design aspect of the project and relative concepts which will be used as a reference for the implementation of the entire project.

The project is separated into three development stages allocated for each section of the project. Prior to the design development of the project, we assume the following project requirements:

- Each sensor nodes are to be placed in different rooms of a house.
- Two rooms require wireless control capabilities to control remote device/s.
- A web interface to display environmental conditions for each location with an interactive interface to control remote devices.

3.1 Sensor Networks

As mentioned in the objectives, we aim to develop a network of wireless sensors that collects environmental information such as temperature, humidity and CO2 of a specific location as well as control a remote device. The location is flexible and can be placed anywhere in the house and the only prerequisite is that sensor nodes are no further than 50 meters apart to maintain the connectivity.

The design approach for our sensor network is of a low-level based approach applying node-level principles where node behaviour runs independently from its neighboring designated nodes, except the base station. While we treat the whole sensor network as an entire network, the failure of one designated node does not affect the present behaviour of the network.

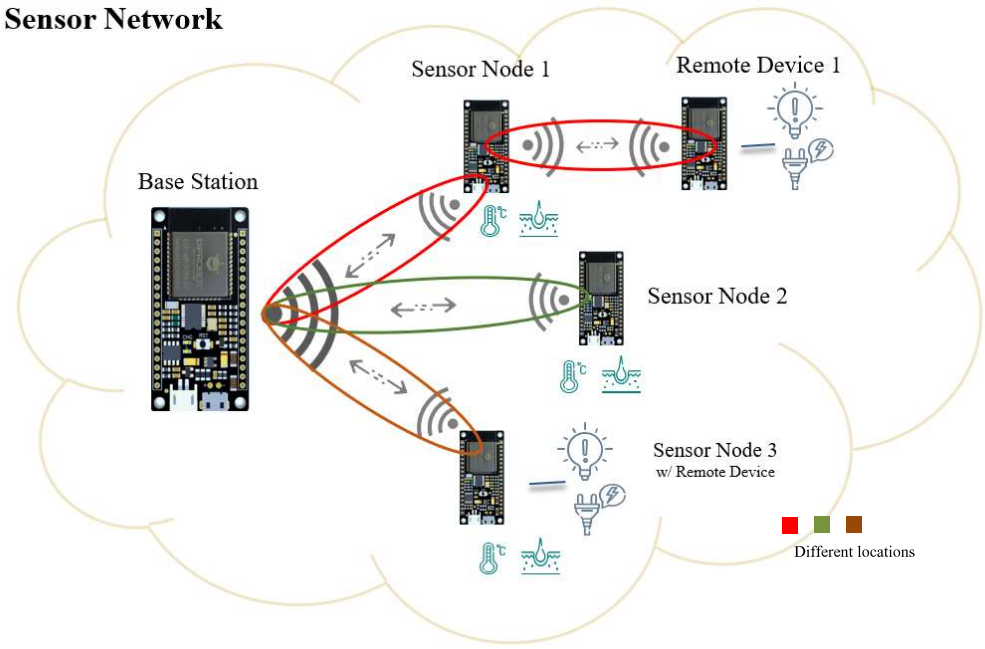

Figure 10: Wireless Sensor Network - Prototype

The designed prototype shown in Figure 10, explains how the intended sensor network is to behave. Each sensor node is presumably placed in different parts of the room with attached environmental sensing capabilities directly communicating to the base station, wirelessly. When new data becomes available, the base stations communicate wirelessly to each designated sensor node individually to pass appropriate instructions received from the webserver.

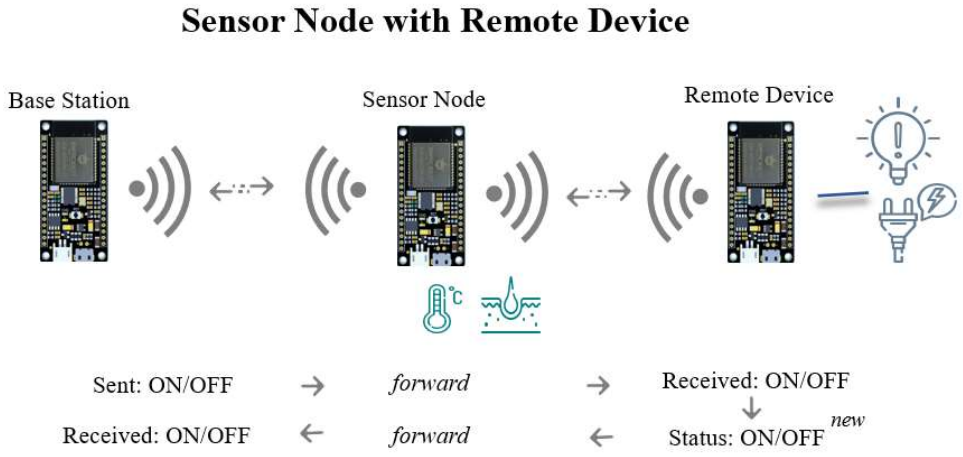

Figure 11: Base Station to Remote Device

The image depicted in Figure 11 shows the behaviour of the data containing the instruction for the remote device. As the data becomes available, the base station passes the information to the sensor node, to which the node will forward the received instruction to a remote device. Similarly, as the new data is received and executed by the remote device, it will send the update back to the sensor node confirming of the changes and the node will forward the new information back to the base station along with the sensor data.

The base station manages all the data received and communicates with the sensor nodes asynchronously. This allows communication between available nodes and the base station with no interruption and for nodes to pass information to the station unconditionally as soon as new data becomes available.

There are two variations with relation to data transmission timing between sensor nodes and the base station. Once of which are for the relative sensor data to be transmitted at every 5-second interval for each sensor node and the other, nodes are to instantaneously transmit data as soon as remote device status has changed.

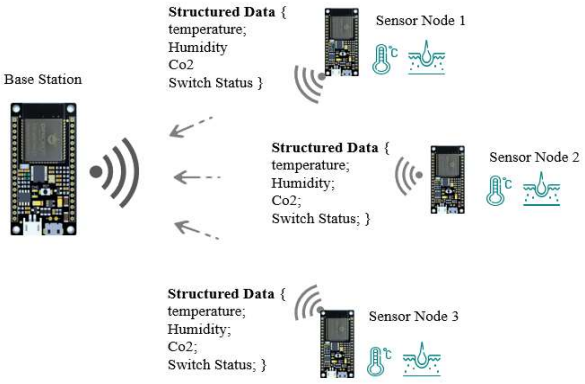

Figure 12: WSN Sensor Data Format

The communication between the base station and the webserver is carried out via a serial interface. Whilst multiple data are received by the base station from the nodes, sent data are to be in a uniformed format and structure to the rest of the sensor nodes, as depicted in Figure 12.

3.2 Web Server

As previously stated within this document, the webserver will be used for two purposes, data handling from the sensors network and handling web client requests, see figure 13.

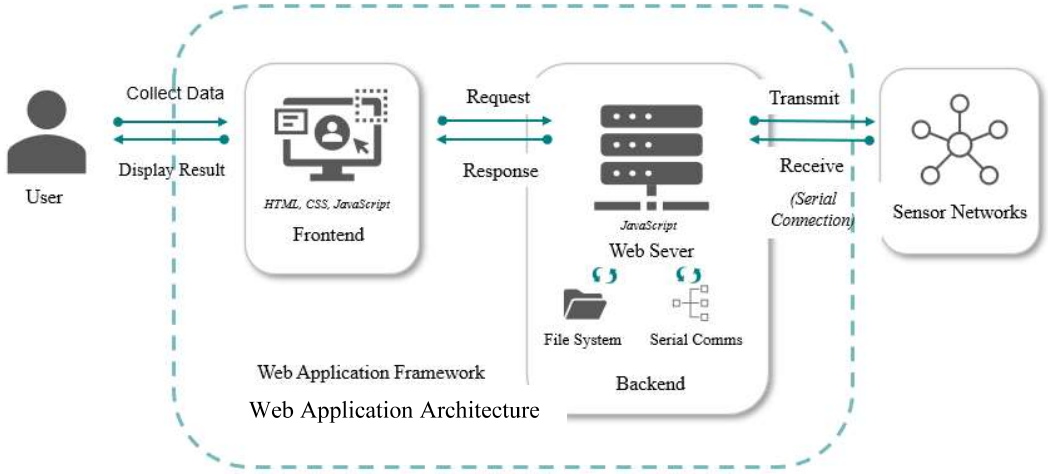

Figure 13: Web Application Architecture

The web server is enabled to proactively listen to a predefined communication port for any transmitted data via serial connection. The received data are assumed to be in a particular format and thus, data checking, error handling and conditions are in place to ensure correct data are received. When such conditions are met, transmitted data are then parsed to which specific data are extracted from its original form, for further analysis. The received data are prepared and where necessary, formatted to satisfy system requirements for data to be displayed on a web page.

As the server is required to respond to HTTP requests and asynchronously publish a stream of sensor data to a page as soon as it became available, we need to enforce additional web socket technology to push new updates to the web interface without refreshing the page. We needed to this due to limitation that came with the standard HTTP protocol i.e. When a response is served to a client following a request, the server-client channel is closed after service is served. The web socket technology used is known as '**Socket.io**' which is a web socket library that enables real-time and bidirectional communication between client and servers. Socket.io provides full-duplex communication channels over a single TCP connection, highly suitable for our application.

3.3 Web Interface

When a request from a client to the webserver is generated, it is expected to return a navigable web page that allows the user to interact with the wireless sensor networks. The information below elaborates the concept design of the web interface, how the interacts with the webserver and ultimately, how it will be implemented.

The design for the web interface is to be kept minimal and simple. To optimize the development and visual look of the web interface, the development will follow the standard and modern way by using HTML, CSS and JavaScript.

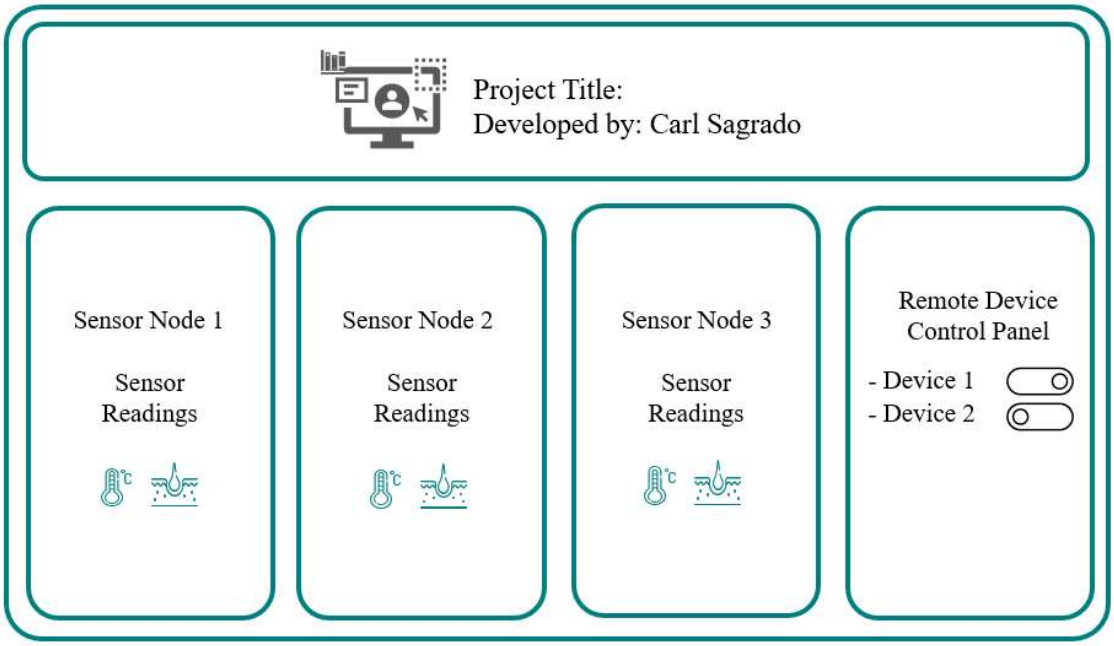

Figure 14: Web Interface Structure

Figure 14 shows the conceptualised design of the web interface. It displays all the information to be displayed, i.e. sensor readings such as temperature, humidity, CO2 as well as the remote device statuses. As intended, streams of data are displayed as soon as it becomes available and received by the webserver without the need of refreshing the page.

As the user interacts with one of the switches in the remote device panel, the generated information by the page will be emitted to the webserver to which will be forwarded serially to the base station, instantaneously. Information immitted should include the location of the remote device and the instruction to which determines the new condition of a remote device. New information sent back by the remote device would be of similar format, processed and displayed accordingly. To enable such feature, programs written in JavaScript will be running on the background of the page which actively listens to new instructions generated via user interaction and new data immitted by the server using via socket.io module.

Chapter 4 - Implementation

This chapter covers the implementation of the project, each part of the project is developed and implemented with reference to the concepts discussed in the design section of this report and in accordance with the objectives outlined.

4.1 Sensor Networks –

While each sensor node will be programmed similar to other sensor nodes, we take into account the addition of remote device feature which relies on the nodes to communicate to the user. To implement the conceptualized design of the overall sensor network, this document discusses the development of each sensor network participant that generally makes up the sensor network as a whole. In general, each node in the network is programmed to uses the ESP-NOW library. [*see **ESP-NOW** in Chapter 1 for more information*].

Wireless Connectivity using ESP-NOW

For implementing wireless connectivity for each node in the sensor network, we use the ESP-NOW communication protocol which utilizes the hardware address for each node, also known as MAC address, to communicate between nodes in the network. Whilst the network is masked and constructed with multiple nodes separated at different distances at different locations, the network uses a peer-to-peer communication method where each node is considered a peer and to communicate, a pairing mechanism are to be initialized.

Whilst there is no handshake performed when establishing connectivity between two network devices, both receiver and sender are set to recognize each other by scanning the peer channel for corresponding mac addresses available on the network. Once the appropriate peers are detected in the network, the mac address of the receiver and the channel id is stored to a special to a memory.

The data sent over the network is set to be in a structured format and requires both the sender and the receiver to have the same structure. To send and receive data requires the use of multiple callback methods to enable successful transmission and receiving of data.

When sending the data over the network, `esp_now_send` function is used, which requires parameters to be passed such as the MAC Address of the receiver, structured data, structured data size. When data is sent over, a callback function is executed which checks the status of the delivery status of the transmission.

To receive data from peers, a callback function called `OnDataRecv` which receives the data and other parameters sent over the network. As previously stated, to successfully receive the

transmitted data, the receiving structure should be of the same format and structure to the sender, otherwise, the data sent are discarded and results to transmission inconsistency.

4.1.1 Node Sensor

To build the sensor node, we follow the physical layout depicted in figure 15. The pins of the DHT module are connected to the ESP32 board as follows:

- **Output Pin** (pin 1) is connected to the pin 21 (GPIO21) of the ESP32 board.
- **VCC Pin (**pin 2) is connected to the 3.3V pin of the board.
- **GND Pin** (pin 3) is connected to the ground pin of the ESP32 board.

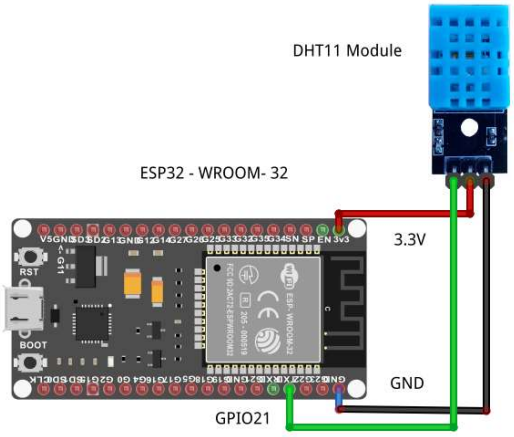

Figure 15: Sensor Node Physical Layout

To program the sensor nodes, we start by including required libraries to enable sensor node to function. The next line is to define the DHT module type for which we use for sensing conditions of an indoor environment and create a DHT object with pin and sensor type.

```
/*Import libraries */
#include <esp_now.h>          //ESP-NOW library for wireless comms
#include <WiFi.h>             //Wi-Fi library
#include <TaskScheduler.h>    //Library for periodic task execution
#include "DHT.h"              //Library for DHT module

#define DHTTYPE DHT11       // Selecting module type DHT 11 or DHT 22
#define DHTPIN 22           //DHT Pin
DHT dht(DHTPIN, DHTTYPE);  //Create a DHT object
```

Under the `setup() function`, we initialized and setting the baud rate of the development board to 115200 bps and started the sensor reading by calling the `dht.begin()` function. The actual readings of each sensor are carried out in an infinite loop, inside the `loop() function`, which uses an in-built function of the DHT library, the values returned by sensor functions are then assigned to float variables, as seen in the code below.

```
  float t = dht.readTemperature();       //read temperature (Celsius)
  float h = dht.readHumidity();   //read humidity (%)
```

The codes above are the relevant key features in enable sensor reading. The next part is preparing the sensor data to be sent over the network. We take into account that some of the sensor nodes also act as a 'forwarder' of data for and from the remote device. Thus, by default, sensor nodes are configured and ready to establish bilateral communication between multiple

boards.

The following codes listed are generally applicable to be used to program each sensor node with a few details that require modification such as receiver's MAC Addresses and relevant variable components that make up a sensor node. Nonetheless, the code snippets below will be of a generally applicable code. And where relevant, specific code for a particular purpose will be discussed.

The structures are created to store sensor readings and to store the incoming switch status.

```
//Structure to store the readings
typedef struct struct_message {
  float temp;
  float hum;
  String device;
  int LEDstate;
} struct_message;

//Structure to store Switch status
typedef struct switchStat {
  int LED;
} switchStat;

struct_message readings;   //Structure to send to base station
switchStat newStatus;      //Structure received from base station
switchStat outgoing;              //Structure to send from remote device
switchStat currentStatus;  //Structure received from remote device
```

The next lines are to create an object of the structure which are used to store sensor data and switch status.

- newStatus structure takes hold of the incoming data from the base station which will be processed and forwarded to the remote device inside the outgoing structure. See figure 16.

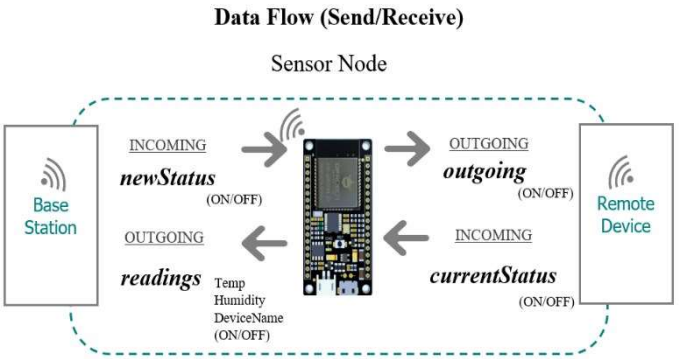

Figure 16: Sensor Node Data Flow

- Incoming data from the remote device is stored inside currentStatus to which will be merged to the sensor readings before the packet is sent to the based station.

When sending data wirelessly, messages are set as tasks. This enables the sensor node to send data periodically with better timing control and power efficiency.

```
//Schedule to send data to base station
Task taskSendMessage( TASK_SECOND * 5, TASK_FOREVER, []() {
  readings.temp = t; //store temp
  readings.hum = h; //store humidity
  readings.device = deviceN; //device name
  readings.LEDstate = state; //remote device state
  esp_err_t result = esp_now_send(receiverMAC, (uint8_t *)
        &readings, sizeof(readings)); //Send msg
}); // start with a 5 second interval

//Schedule to send data to remote
Task taskSendMessage1( TASK_SECOND * 0.5, TASK_FOREVER, []() { device
  outgoing.LED = newState;
  esp_err_t result = esp_now_send(receiverMAC1,(uint8_t *)
        &outgoing, sizeof(outgoing));//Send msg
}); // start with a 500 milli second interval
```

As per the design, there are two variations in transmission timing. Messages sent to the base station are set to be transmitted every 5-second interval and messages sent to the remote are scheduled as soon as new data is received and invoked to be transmitted after every 500 milliseconds, as seen in the code snippet above.

```
// Callback when data is received
void OnDataRecv(const uint8_t * mac_addr, const uint8_t *incomingData, int len) {
  char macStr[18]; // Read in MAC address to buffer
  snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x",
           mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4], mac_addr[5]);
 String x = mac_addr[0] + mac_addr[1] + mac_addr[2] + mac_addr[3] + mac_addr[4] + mac_addr[5
];

  if (x == "447") { //from the BS [with MAC ADDRESS: 10:52:1C:64:69:74] - converted
    memcpy(&newStatus, incomingData, sizeof(newStatus));
    newState = newStatus.LED; //store the new status
    Serial.println("FROM MCU -" );
     userScheduler1.execute();
  }
  if (x == "969") { //from remote device [with MAC ADDRESS: F0:08:D1:D3:3D:F0] - converted
    memcpy(&currentStatus, incomingData, sizeof(currentStatus));
      state = currentStatus.LED;
     Serial.print("FROM RD - LED State: ");
     Serial.print(state);
  }
```

To differentiate data from multiple senders, we use the sender's MAC address information and use it as a parameter to check its availability. For a sensor node, incoming data received from the base station triggers an event that pushes a task that forwards received data to a wirelessly connected remote device, to queue for sending as seen above snippet of the onDataReceive callback function. Complete code in Appendix II.

4.1.2 Remote Device

The remote devices are built using ESP32 and a relay switch. Development of the end node is based on the physical layout depicted in figure 17 and 18. For practical implementation, we use an LED lamp as a device for which system will remotely control. The device can be anything and follows the same circuitry as below.

We use Figure 17 as a reference for the pinouts where the pins of the relay module are connected to the module as follows:

- **Input Pin** is connected to the pin 22 (GPIO22) of the ESP32 board.
- **VCC Pin** is connected to the VCC pin of the board.
- **GND Pin** is connected to the ground pin of the ESP32 board.
- **Normally Open (NO) Pin** is connected to the Anode (+) of the Led.
- **Common Contact Pin** is connected to the VCC pin of the board.

Note: For practical applications, relay module requires 5V supply to be fully operational. Schematic shown before is for demonstration purpose only.

The end node incorporates a push-button to allow users to manually control the state of a remote device. The pin configuration has one pin of the switch connected to a power source and another pin connected to the ground. To allow detection of the button status, an opposite pin is connected to the pin 22 (GPIO22) of the ESP32 board.

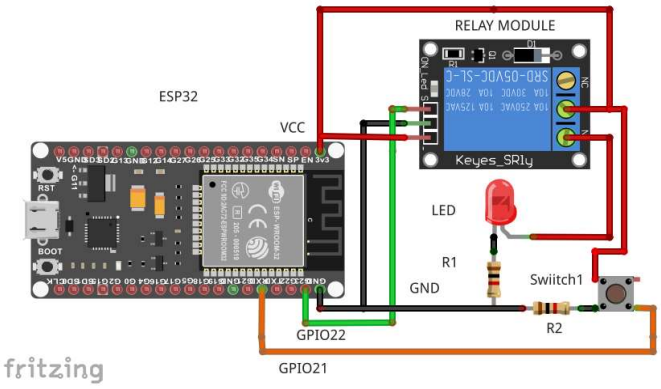

Figure 18: End Node Physical Layout

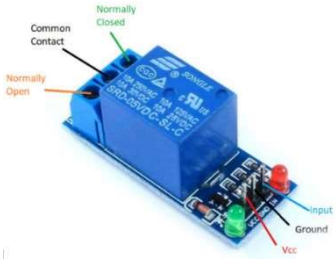

Figure 17: Relay Module Pinout Reference

The program for end devices requires the inclusion of necessary libraries to enable a device to communicate wirelessly, exactly the sensor node. Seen in the code below, two GPIO pins are used. One of which is for sending a signal to the relay and the other for reading the condition

```
/*Import libraries */
#include <esp_now.h>         //ESP-NOW library for wireless comms
#include <WiFi.h>            //Wi-Fi library
#include <TaskScheduler.h>   //Library for periodic task execution

#define Relay 22     // Using pin 22 for relay signal
#define switch 21     // Using pin 21 to read the switch
```

To correctly detect the status of the push-button, the pin 22 as an input. The digital signal outputted by the pin will be stored in a variable and a condition is set for each time changes to the button is made, as seen in the code below.

```
void loop() {
  buttonNew = digitalRead(switch); //read the input pin
//Check if button is pressed
  if ((buttonOld == 0 && buttonNew == 1)) {
    if (LEDState == 0) { //If 'OFF'
      digitalWrite(Relay, HIGH); //turn the device  'ON'
      LEDState = 1;
      delay(1000); //debounce delay
    } else {
      digitalWrite(Relay, LOW);
      LEDState = 0;
      delay(1000); //debounce
    }
    userScheduler.execute(); //Send device update
  }
  buttonOld = buttonNew;
}
```

When push-button is pushed, signal input is assigned to a variable which holds the current condition of the relay. The current value is inverted, and the new relay output is also reversed.

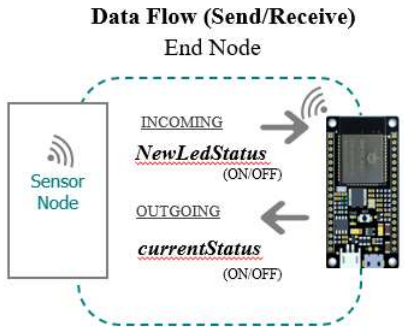

Figure 19: End Node Data Flow

The next part is preparing the remote device data to be sent over the network. Similar configuration to the sensor node, by default, the end nodes are set and configured to establish bilateral communication to allow wireless transmission for sending and receiving of data.

Depicted in Figure 19, the data structure of incoming and outgoing data is identical format structure to enable data flow consistency.

Seen below is the declaration code of the structure for storing incoming and outgoing data.

```
//structure to device instruction
typedef struct LEDstate {
  int LEDstatus;
} LEDstate;

LEDstate newLedStatus;  //structure used for incoming data
LEDstate currentStatus; //structure used for outgoing data
```

For sending data, we call a specific task which executes the necessary data collection and assignment to the outgoing structure data, set to transmit after 500 milliseconds. When data is sent, a callback function is called to check the state of the message transmission, see code below.

```
//Schedule to send data to sensor node
Task taskSendMessage( TASK_SECOND * 0.5, TASK_FOREVER, []() {
    currentStatus.LEDstatus = LEDnewState; //store led status
  esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *)&currentStatus,
 sizeof(currentStatus)); // Send message via ESP-NOW

 //check if msg transmission status
  if (result == ESP_OK) {
    Serial.println("Sent with success");
  }
  else {
    Serial.println("Error sending the data");
  }
});
```

There are 2 instances for which task scheduler function is called to transmit data, the function execution occurs after one of the two events are met satisfied, which are:

- Post instruction execution – After received data is executed, a status update is to be sent.
- Post manual switch trigger – Changes applied after manual control of the device. A status update is to be sent.

```
// Callback when data is received
void OnDataRecv(const uint8_t * mac_addr, const uint8_t *incomingData, int len
) {
  memcpy(&newLedStatus, incomingData, sizeof(newLedStatus));
  LEDState = newLedStatus.LEDstatus; //store the new status

   Serial.print("Data Received: - LED: "); //store received status
   Serial.println(LEDState);
}
```

To differentiate data from multiple senders, we use the sender's MAC address information and use it as a parameter to check its availability. For a sensor node, incoming data received from the base station triggers an event that pushes a task that forwards received data to a wirelessly connected remote device, to queue for sending as seen above snippet of the onDataReceive callback function.

4.1.2 Base Station:

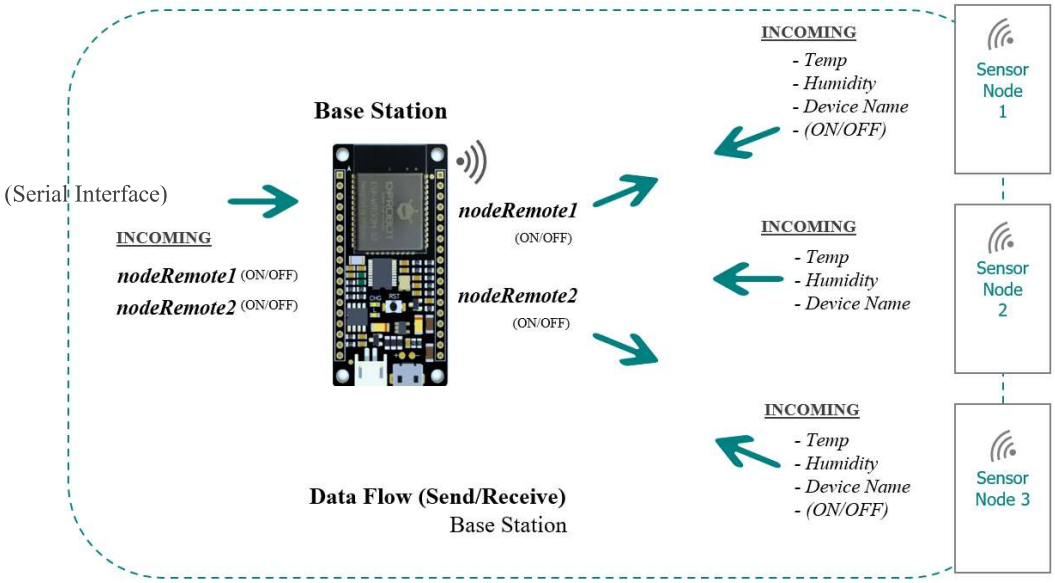

Figure 20: Base Station Data Flow

Developing the base station require considerations relevant to managing the incoming and outgoing of data.

- Serial Interface Communication – With reference to the design, the serial interface is the communication interface used to communicate the sensor network and the web server. This media is used by the server to send new instructions to the remote device and to receive sensor data.
- Sensor Network Gateway - The Base Station acts as a gateway for the sensor network to transmit sensor and remote device data and to receive new instructions from the user interface for the remote device.

Both topics are a prerequisite to the development of the base station as we require for communication mediums to work under a single module, seamlessly. As data transmitted over a medium is relative to the transmitted data over the other medium, therefore, requires data processing and management, see figure 20.

```
/*Import libraries */
#include <esp_now.h> //ESP-NOW library for wireless comms
#include <WiFi.h> //Wi-Fi library
#include <ArduinoJson.h> //required for converting data to json format
#include <TaskScheduler.h> //Library for periodic task execution
```

The code above shows the necessary libraries for the base station to work.

The base station maintains two data structures which is used for storing serial data and wireless data, respectively. The structure of the code can be seen overleaf.

```
//structure for serial data
typedef struct switchStat {
  int LED;
} switchStat;

//structure for wireless data
typedef struct struct_message {
  float temp; //temperature
  float hum; //humidity
  String device; // device name
  int LEDstate; //remote device state
} struct_message;
```

Wireless Communication between Sensor Nodes

As the format of transmitted data is structured and identically implemented to all sensor nodes, the system does not require any detection mechanism to determine the sender. Thus, the base station is programmed to treat any incoming data to be treated equally and instantaneously processed.

```
void OnDataRecv(const uint8_t * mac_addr, const uint8_t *incomingData, int len) {
  char macStr[18];
  snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x",
           mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4], mac_ad
dr[5]);
  Serial.println(macStr);
  memcpy(&incomingReadings, incomingData, sizeof(incomingReadings));
  incomingTemp = incomingReadings.temp; //readings for temperature
  incomingHum = incomingReadings.hum; //readings for humidity
  deviceName = incomingReadings.device; //device name
  ledStateR = incomingReadings.LEDstate; //remote device status
  receviedData(); //function call to transmit received data to serial.
}
```

When sending new data to an appropriate sensor node, a task scheduler is called to perform the operation. The code structure of the function is similar to what was previously used and can be seen below.

```
Task taskSendMessage( TASK_SECOND * 0.5, TASK_FOREVER, []() {
  esp_err_t result = esp_now_send(macAddress, (uint8_t *)&nodeRD1, sizeof(nodeRD1)
); // Send message via ESP-NOW
}); // start with a 500ms interval

Task taskSendMessage1( TASK_SECOND * 0.5, TASK_FOREVER, []() {
  esp_err_t result = esp_now_send(macAddress1, (uint8_t *)&nodeRD2, sizeof(nodeRD2
)); // Send message via ESP-NOW
}); // start with a 500ms interval
```

Each sendMessage is called relative to the new data received via serial port with an execution time set to 500milliseconds.

Serial Communication – Base Station and Server

The baud rate of the base station for serial data transmission is set to *115200 bps* which is set uniformly with the web server.

To receive incoming data via serial communication, the serial port is continuously checked for any data being transmitted. The code below shows how the serial is detected by the base station.

```
if (Serial.available()) {    // Check if any data is available
    command = Serial.readStringUntil('\n'); // read the data and store a var
    if (command.equals("R1ON")) { // R1ON = Turn on Device 1
      nodeRD1.LED = 1;
      userScheduler.execute(); // R1ON = Turn ON Device 1
    } else if (command.equals("R1OFF")) { // R1ON = Turn OFF Device 1
      nodeRD1.LED = 0;
      userScheduler.execute();    //call task scheduler to send the message
    } else if (command.equals("R2ON")) { // R2ON = Turn OFF Device 2
      nodeRD2.LED = 1;
      userScheduler1.execute();
    } else if (command.equals("R2OFF")) { // R1ON = Turn OFF Device 2
      nodeRD2.LED = 0;
      userScheduler1.execute(); //call task scheduler to send the message
    }
}
```

The received data via wireless are converted into a JSON format to allow a data parsing at the server-side. **JSON** (JavaScript Object Notation) is a data representation format that modern web server uses to transfer and access structured data.

```
void receviedData() {

  String msg2serial;
  DynamicJsonDocument message(1024); //create JSON formatted document

  message["LEDState"] = ledStateR; //assign end device value
  message["TEMP"] = incomingTemp ; //assign Temp data
  message["HUM"] = incomingHum;    //assign humidity data
  message["deviceName"] = deviceName;

  serializeJson(message, msg2serial);  //Serialize JSON document
  Serial.println(msg2serial);
}
```

The snippet above shows the function which performs the formatting of the received data to a JSON format before transmitting the data to the web server through a serial interface.

4.2 Web Server

We use a Node.js web application server framework called 'Express.js' as a framework for developing the server which allows managing of connections between the web client and the server allowing the server to respond to HTTP requests. The framework also defines the routing table for which performs actions based on the HTTP method and URL.

The next line of codes below is applied to set up the express server. The server listens to requests on port *4000.* A callback function is called which logs a message to indicate that the server is actively listening for any request.

To serve a web page to the browser, a middleware function `use()`, is used to direct the path of express to point the folder to serve.

```
var express = require("express"); //require express app

//Express app setup
var app = express(); //invoke express function
var http = require('http').Server(app);

app.use(express.static('public')); //express app to server the public folder

var listenFunction = http.listen(4000, function () { //listen requests on port 4000
  console.log('listening on *:4000');
});
```

To use socket.io to attain full-duplex communication between client and server, we use a variable which holds the required socket.io web socket, which will be used as reference to call methods.

```
var io = require('socket.io')(http); //Setup socket.io to serve the server
```

Callback function below listens for an event called 'connection' for when a connection is made between the server and the client. To listen for an event using the *on* method, the event name and callback function are passed as arguments. The passed callback parameter refers to the instance of the socket representing the connection between the client and the server.

```
io.on('connection', function (socket) {
    socket.on('disconnect', function () {
      activeClients -= 1; //Decrement when a socket is disconnected
      io.emit('nClientUpdate', { clients: activeClients, clientLogger: state });
    });   //Emit to web page

    activeClients += 1;  //Increment when a socket is disconnected
    io.emit('nClientUpdate', { clients: activeClients, clientLogger: state });
          //Emit to web page
      }
```

To seamlessly transmit data to the client from the web server using a single connection and vice versa, the `emit` method is used and applied for both client and the server which allows the use of custom events to be emitted to the client. This provides convenience in handling data.

Following the import of the Serialport module, the module library is referenced, where an instance of the module is created with the port name and the baud rate are passed as parameters. The baud rate value is set identical to the baud rate for which the base station is set at. When the specified port is open, the module provides two ways of reading the data, one of which is by reading the data as a raw data itself and the other is reading through formatting the raw data. Incoming data is regarded as raw data and to extract the necessary information from its current format, raw data is passed and parsed into a variable which will the data into a single string with a newline character for each line. The string data is parsed using `JSON.parse()` function, converting data to a JavaScript object for which will be emitted and presented on the web page. The code snippet below depicts a simple process of how programming instructions were developed on the server with the Serialport module, from importing the module, data parsing, up to emitting the data to be used by the frontend.

```
var SerialPort = require('serialport'); //import SerialPort module
var Readline = require('@serialport/parser-readline');

var connect2Serial = function () {
   var parser = port.pipe(new Readline());  // seperate data input using newlines
    parser.on('data', function (data) {
      var msgData = JSON.parse(data); // parsing JSON string data
      if (msgData.deviceName == "Room 2") { //"Room 2 data"
       //  emit data readings with reviver to web page
       io.emit('R2TEMPState', { R2temp: r2temp }); //R2 temperature
       io.emit('R2HUMState', { R2humidity: r2hum });//R2 humidity
       io.emit('R2ActiveDeviceName', { R2active: r2device }); //R2 device name
       io.emit('R2DeviceState', { R2DeviceStat: R2deviceStat }); //R2 device status
       io.emit('R2LEDState', { R2LED: r2LEDstate }); //R2 Remote Device status
       io.emit('DeviceCount', { DeviceCount: r2activeDevice }); //R2 device count
      }
      if (msgData.deviceName == "Room 1") { //"Room 1 data"
       //  emit data readings with reviver to web page
       io.emit('R1TEMPState', { R1temp: r1temp }); //R1 temperature
       io.emit('R1HUMState', { R1humidity: r1hum });//R1 humidity
       io.emit('R1ActiveDeviceName', { R1active: r1device }); //R1 device name
       io.emit('R1DeviceState', { R1DeviceStat: R1deviceStat }); //R1 device status
       io.emit('R1LEDState', { R1LED: r1LEDstate }); //R1 Remote Device status
       io.emit('DeviceCount', { DeviceCount: r1activeDevice }); //R1 device count
      }
   }
```

4.3 Web Interface

The development of the web interface is based on the concept design in Section 3 of this report. Seen in figure 21 below, depicts the concept design developed into an actual web interface prototype of the WSN network. The web interface was given a basic structure using HTML where each element is provided with identifiable attributes which is used for styling with CSS and adding functionalities to the web page, as discussed below.

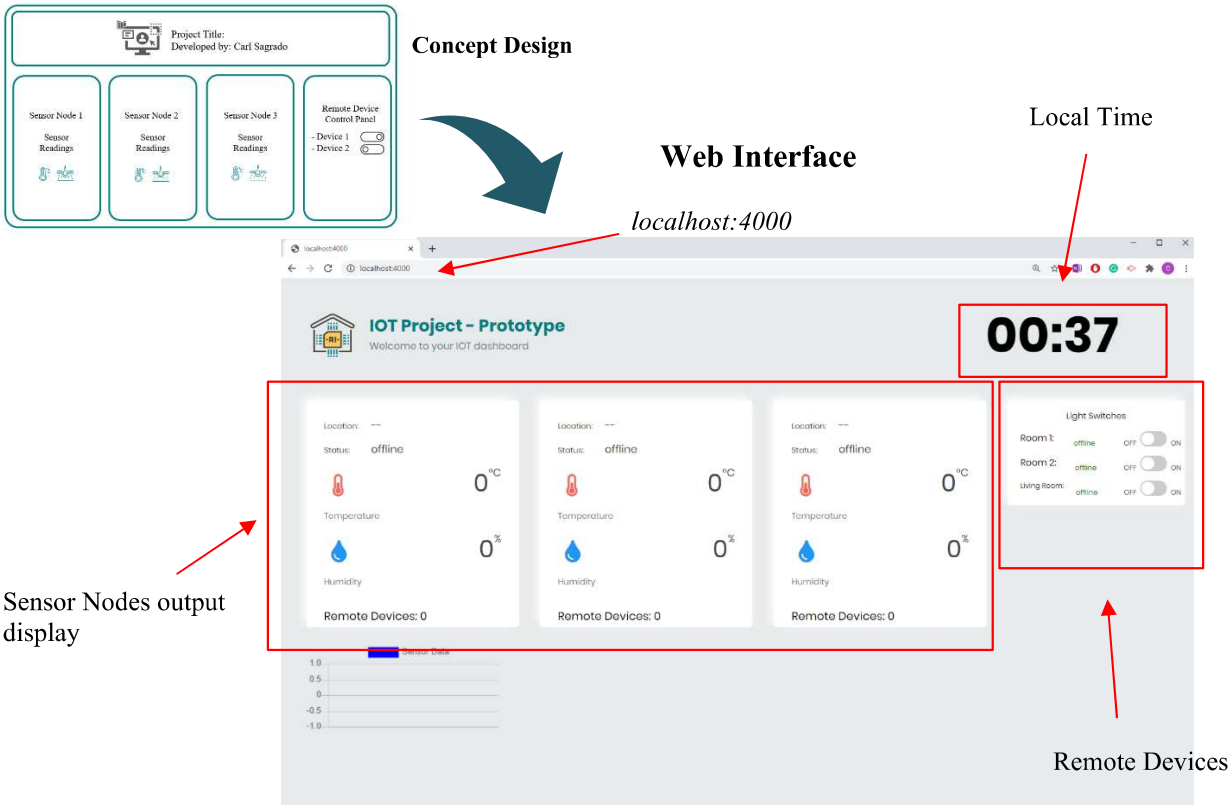

Figure 21: Web Interface Implementation

The underlying functions of the page were written in JavaScript, and its embedded ability to asynchronously connect to the web server is through a web socket connection.

Using JavaScript, HTML element contents are navigated through its corresponding DOM reference [reference] where each element are represented as objects and accessed through its attributes to perform manipulations such as displaying information to the relevant elements of the page. When a socket connection is made, the WSN web interface is loaded and real-time sensor data is displayed. Emitted data from the server is received and detected using a callback function which listens to specific events. Using the `on` method, named events are passed with a callback function as an argument corresponding to the data emitted by the server. Conversely, data generated from the user interface is emitted to the server using the `emit` function, implemented in the web server.

4.3.1 Remote Devices –

The end devices are remotely controlled through the switches on the interface. When switches are toggled, each individual switch emits specific data to the web server which is read, processed, and transmitted to base station through the serial interface. As a WSN controller, the base station processes the received data and extracts the new instructions which includes designated node device and switch command. The information is sent to the sensor node which acts as a forwarder and forwards relevant data to the end device for execution.

Instantaneously, newly received instructions are executed at the end device to which, in effect, generates a new update correlating to the status of the device following execution. The status update is transmitted back to the sensor node, to which the node detects and uses the received values as the end device data of its sensor data structure which is sent to the web server via the base station. The returned status update from a specific end device causes the toggle effect to remain in its position i.e., from on position to off and vice versa. When peer connection is lost between the sensor node and the end node, the end node will transmit the default data indicating the current switch status. This in turn will be transmitted and used as a default switch data to which will be displayed on the page when a new socket connection is established. The server will constantly stream the data received as it arrives as such, when the web page is fully loaded, position of the switch status position will be based on the relative data that is available on the server.

4.3.2 User perspective –

When a user enters the correct web address to the URL of the web browser, a web page will be served and served and displayed on the browser. Of which, where relatively middle of the page, the user can observe a section where it shows some information. The first three columns display the data collected from each node in the wireless sensor network. Such data includes location of the node, node status, sensor readings and the number of devices that are connected to one sensor node. These data are real-time data collected from the sensor network through the web server. The last column that can be observed is the light switch section where toggle switches are available. These switches can be toggled, allowing the user to remote device to turn on or off. When toggled, the switches display a transition animation between on and off, providing better user experience of understanding the events occurring underneath the interface, visually.

Figure 22 and figure 23 shows the indoor infrastructure placement location of each node and the physical layout of the project, respectively.

Figure 22: Indoor infrastructure and node's location

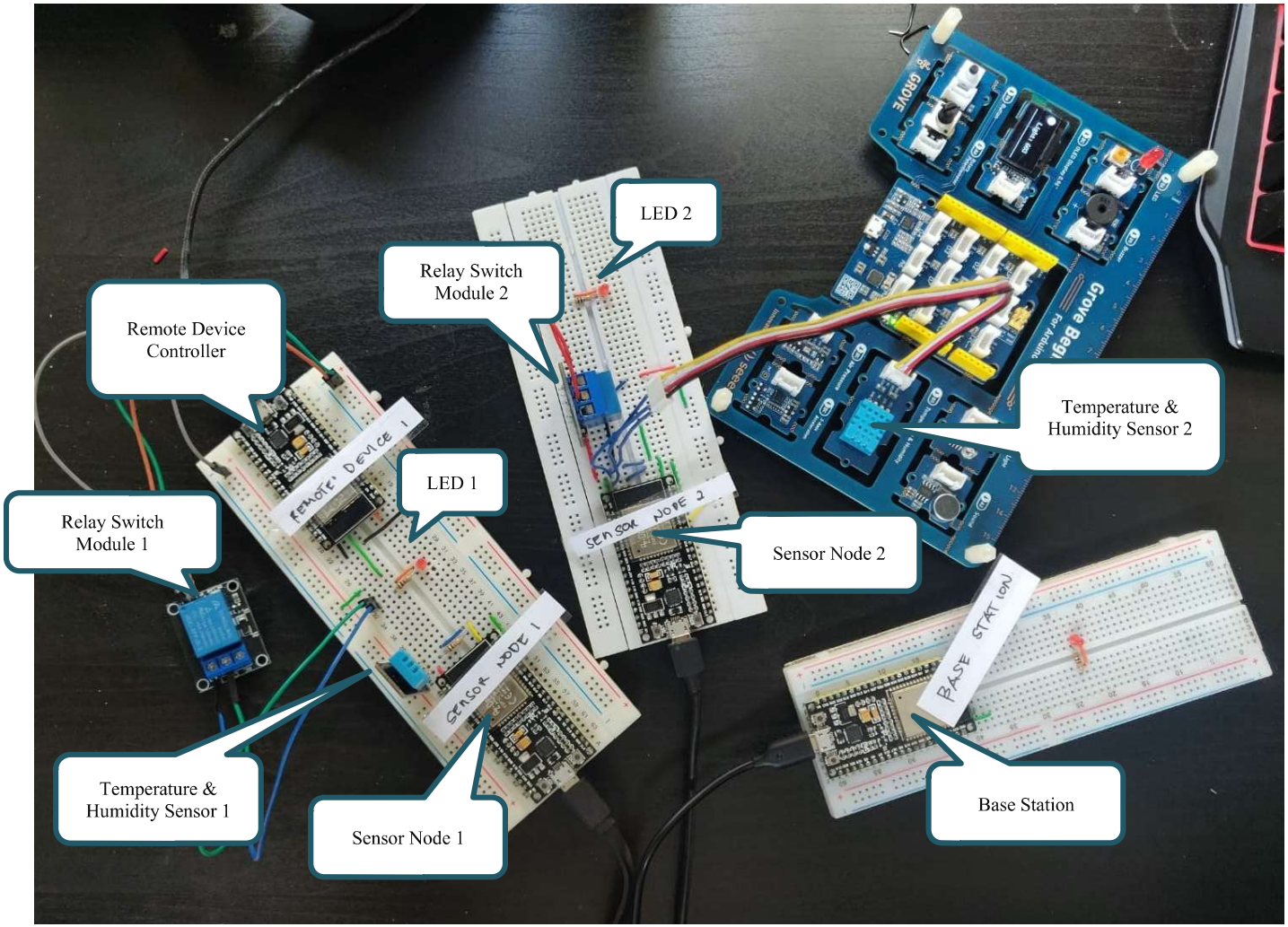

Figure 23: Physical Product Layout

Chapter 5 – Requirements/Bill of Materials

Below is the table for the requirements needed to build the project prototype.

Table 2: Bill of Materials

SCHID	Component	Supplier	Quantity	Price
Sensor Node1, Sensor Node 2, Remote Device 1, Remote Device 2, Base Station	ESP32 – IoT Board	Farnell	5	**€15.95**
Relay Switch 1 Relay Switch 2	Relay Module	RS	2	**€5.84**
R1, R2, R3	RESISTOR, 1kΩ	Farnell	3	**€0.17**
R4, R5, R6	RESISTOR, 10kΩ	Farnell	3	**€0.13**
SD1, SD2, SD3	DHT11 (Sensor Modules)	RS	3	**€5.25**
CO2S1	CO2 Sensor	RS	3	**€36.39**
			Total	**201.5**

**Note: The total cost listed is only the estimated price based on two of the electronic supplier's website listing. Additional cost may incur for the future improvement of the project. The sensors listed are not dependent to each other and thus, the entire project can still provide some functionalities with or without the other sensor.*

Chapter 6 – Future Work

While the wireless sensor network prototype is fully functional, the current prototype system only displays 2 relevant sensor data. It would be worth adding additional environmental data such as CO2, smoke detection, airflow system, etc. The system could also improve by utilising the home wireless router to enable wireless communication between the base station and the web server, which helps remove the need serial cable. This allows placement flexibility of the base station enabling a full wireless system. Using proper router allows the project to scale to other avenues such as adding a voice command to allow users to control wireless end devices through voice without physical interaction. The web page could improve by configuring the page to be responsive, allowing mobile devices to have access and interact to the page.

An additional work that can be done to the prototype is to upgrade its current network setup to a be a more self-organizing network. We define a self-organizing network as a network of nodes, where its nodes coordinate with each other to form a system that adapts to network changes to achieve its goal more efficiently. The main work in modifying the current network setup involves the configuring of the web server, the base station node, the sensor nodes and the remote nodes. The concept is to store each nodes' MAC Addresses and its role to the web server. The new setup would allow the base station node to decide which node owns the designated role based on the information sent by web server. The web server would periodically transmit the addresses to the base station, and the base station will periodically perform a handshake protocol to confirm the availability status of each node. If one of the nodes becomes inactive (i.e., shuts down, breaks, or for unknown reason stops communicating to the network), the base station would search the network to determine which node is the closest to the malfunctioning node, as well as determine which network node has been affected. When the search completes, affected nodes will receive a new node address and prompts for new pairing and where necessary, updates all other nodes for the new changes.

The added benefit of storing the MAC addresses and other relevant information of a network node to the web server is that it gives way to other possible improvements and other features to the current system. For example, since the web server hosts such key information about the individual units of the network, a feature can be added in the interface to enable users to manually add and remove new network nodes to the system.

Chapter 7 - Conclusion

The aim of this project was to design a wireless sensor network for home automation. The project was to provide a system platform that reads environmental of a residential unit and publishes the sensor data in real-time in a web interface with control capabilities enabling users to control remote devices wireless using the same interface.

The whole system architecture was presented in detail where each component of the project such sensor node, web server, and web interface were discussed. The background of different technologies and other elements of the project was discussed in chapter 2 of this report while the design and implementation of the prototype was presented in chapter 3 and 4, respectively.

Overall, the core objective of the project was met, and the result provided us with a working prototype system, thus, concludes this project.

Chapter 8 - Engineering implications

8.1. Environmental Issues Concerning Electronic Assembly

"The Electronics industry has far transformed the facet of society. In addition to providing the basis for the information revolution, electronics enable the society's vital support systems, including those that provide for such necessities as food, water, energy, transportation, health care, telecommunications, trade and finance." [4]
The emerging challenges of the Electronics Industry are its impact on the environment. And thus, the task to tackle this impact has become one of the challenges of the industry since the Electronics sector is by no means slowing down and stopping anytime.

8.2 Production

"The environmental impacts associated with a product include those that occur in the production processes to make it. These impacts occur not only at the final stage of assembling parts but also in the production of parts and their constituent materials." [5]

Consuming non-renewable resources including precious metals like gold are used to make electronic component e.g. Processor. The mining of such resources pollutes the water of surrounding the communities through cyanide contaminated waste ore and other abandoned mine waste including toxic metals and acid are often get released into lakes, streams, and the ocean, killing fish and contaminating drinking water. [6]

Another concern is the production of electronics such as microchips, printed circuit boards and computers is the use of many nasty chemicals during production. The industry has become globally competitive in the face of a rapidly changing technology which greatly increased the demand to produce more of the said products. The production requires a meticulous and tedious process which requires workers to be exposed with the chemicals and acids such as phosphoric acid, hydrofluoric acid, nitric acid, sulfuric ammonia, hydrogen peroxide, isopropyl alcohol, tetramethylammonium hydroxide, acetone, hexamethyldisilane, etc. which are harmful substances [7]and when processed, releases potential air emissions which include: toxic, reactive and hazardous gases that contributes to the rising global warming.

The products used to manufacture and produce technology often contain solvents called Volatile Organic Compounds, or VOC's which are Carbon-based compounds that vaporise easily at a certain temperature. VOC's affects the environment by contributing towards the

formation of ground-level ozone, the main component of smog (an air pollutant) to which does not only have many detrimental effects in the environment, causes health problems to anyone when overexposed. [8]

Due to the alarming effect to global warming, regulations such as REACH (Registration, Evaluation, and Authorisation of Chemicals) were set out and established to have an immediate implementation globally and companies are required to meet the standards.

8.3 Usage

Power consumption and demand has risen due to technology. Though for some, technology has aided humanity to become efficient to their work and even for some allowed them to connect to individuals from far distances. As technology helped humanity get through our primitive beings and allow us to multi-task, the end game of this advancement is the toll to our environment. For example, we always need electric energy to keep our gadgets running, to keep our lights on, etc. Engineers and Developers alike are encouraged to consider the environmental impact of its products and to find ways to reduce possible waste and unnecessary energy consumption when designing new products.

Electric energy is generated by the use of fossil or nuclear fuels on a large scale. New renewable technologies may have been available, however, that too creates and contributes to global warming. The exact type and intensity of environmental impacts may have varied depending on the specific technology used, location, and other factors, however, is still, in fact, contributes to global warming [9].

Another factor arising from the use of the electronic assembly is the heat generated by electronic devices. As the heat dissipates to the environment it creates carbon dioxide (CO2), a gas that stores heat which contributes to greenhouse warming. [10]

To address the issue, a program called ENERGY STAR was developed, a product specification process that relies on rigorous market, engineering and pollution savings analysis to tackle the concurring greenhouse problem. The program test evaluates and validates products to control and reduce greenhouse emissions. The specifications laid out by ENERGY STAR aims to direct consumers to a more energy-efficient product which would lessen the greenhouse gas contribution to greenhouse warming [11].

8.4 Disposal

Electronic assembly disposal has increasingly acute as more devices invented every day rendering previous devices obsolete. These electronic wastes are known as “Techno trash” where any broken or unwanted electrical or electronic device is currently the most rapidly growing type of waste. Most of these wastes contain non-biodegradable materials, heavy metals and toxic materials that end up in a landfill that contributes to the growing problem of land pollution, in addition to land pollution, over time, these toxic wastes end up contaminating the water that animals, humans, plants consume. [12]

8.5 Conclusion

While electronic devices have positively impacted society in a massive way. We must also take account of the negative impact it does to our environment. The government and the environmental organisation all over the world have participated acted towards preserving the environment as much as we can and therefore, everyone is incumbent to participate. There are number of ways we can help the environment, such as; avoid producing any electronic assembly that has too little impact or no significance to the society, monitor waste production, producing products that are energy sufficient, use devices when necessary and turn off when not needed, separate techno trash from ordinary household trash and dispose or donate unwanted electronic devices to be properly recycled.

References

[1] E. Systems, "ESP32 Series Datasheet," 2021. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf. [Accessed 22 01 2021].

[2] S. Santos, "ESP32 Pinout Reference: Which GPIO pins should you use?," Random Nerd Tutorials, 2021. [Online]. Available: https://randomnerdtutorials.com/esp32-pinout-reference-gpios/. [Accessed 21 01 2021].

[3] "CO2 and RH/T Sensor Module," Sensirion, 2021. [Online]. Available: https://www.sensirion.com/en/environmental-sensors/carbon-dioxide-sensors/carbon-dioxide-sensors-co2/#:~:text=SCD30%20%2D%20Sensor%20Module%20for%20HVAC%20and%20Indoor%20Air%20Quality%20Applications&text=Thanks%20to%20the%20dual%2Dchannel,easy%20integrati. [Accessed 22 01 2021].

[4] "Industrial Environmental Performance Metrics," 1999. [Online]. Available: nap.edu/read/9458/chapter/8#108. [Accessed 14 01 2021].

[5] E. Williams, "Environmental Impacts in the Production of Personal Computers," July 2003. [Online]. Available: https://www.researchgate.net/publication/226596315_Environmental_Impacts_in_the_Production_of_Personal_Computers. [Accessed 10 01 2021].

[6] "How to Recycle Your Electricals – any why you should," 27 11 2020. [Online]. Available: https://littlegreenblog.com/green-technology/waste-and-recycling/recycle-your-electricals/. [Accessed 13 01 2021].

[7] "Industrias Chemicals," 2021. [Online]. Available: https://www.linde-gas.com/en/products_and_supply/electronic_gases_and_chemicals/wet_chemicals/index.html. [Accessed 20 01 2021].

[8] H. Mehendale, "Volatile Organic Compounds," 2014. [Online]. Available: https://www.sciencedirect.com/topics/chemistry/volatile-organic-compound. [Accessed 16 01 2021].

[9] "Environmental Impacts of Renewable Energy Technologies," 14 July 2008. [Online]. Available: https://www.ucsusa.org/resources/environmental-impacts-renewable-energy-technologies. [Accessed 17 01 2021].

[10] J. Cook, "Waste heat vs Greenhouse warming," 27 July 2010. [Online]. Available: https://skepticalscience.com/Waste-heat-vs-greenhouse-warming.html. [Accessed 22 01 2021].

[11] "Energy Star," 2021. [Online]. Available: https://ec.europa.eu/energy/topics/energy-efficiency/energy-efficient-products/energy-star_en. [Accessed 19 01 2021].

[12] "ENERGY STAR Product Specification Development Process Description," 2021. [Online]. Available: https://www.energystar.gov/partner_resources/product_specification_development_process#:~:text=There%20are%20certain%20elements%20characteristic,stakeholder%20comment%20on%20each%20draft.. [Accessed 19 01 2021].

[13] "The Internet of Things with ESP32," ESP32.net, [Online]. Available: ESP32.net. [Accessed 21 01 2021].

[14] Syed Zain Nasir, "Introduction to DHT11," The Engineering Projects, [Online]. Available: https://www.theengineeringprojects.com/2019/03/introduction-to-dht11.html. [Accessed 22 01 2021].

[15] "5V Single-Channel Relay Module," Components 101, 21 12 2020. [Online]. Available: https://components101.com/switches/5v-single-channel-relay-module-pinout-features-applications-working-datasheet. [Accessed 22 01 2021].

Appendix I – Remote Device Source Code

```
/*  Title: WSN Project - Remote Device
    Written by: Carl Sagrado
    Student No: X00084403
    Date:       04/05/2021
    Current version:    v1.3

**Version Updates:
    v1.0 - Switch button added to the circuit to control relay.
    v1.1 - Added ESP_NOW for sending and receiving data wirelessly.
    v1.2 - Added Task scheduler as an alternative solution .
            to remove unnecessary use of delay into the function
    v1.3 - Modified to execute received instructions and  to
            automatically send status to after execution

**Desc:
    - This is an end device controller that contols the status a relay switch
 */


#include <esp_now.h> //include ESPOW library for ESP32 to this program
#include <WiFi.h>   //include Wi-Fi library to this program
#include <TaskScheduler.h> //include task scheduler library to this program

#define Relay 22        //Relay pinout
#define switch 21       //Physical switch pinout

//Receiver MAC Address
uint8_t receiver[] = {0x10, 0x52, 0x1C, 0x62, 0xB4, 0x08}; //Sensor Node 1

//data Structure format for switch data
typedef struct LEDstate {
  int LEDstatus; //LED status
} LEDstate;

LEDstate newLedStatus;  //Structure for received switch instruction.
LEDstate currentStatus; //Structure used to send current switch status.

/*Global variables */
int LEDState;           //switch status
int LEDnewState;        //varaible

int buttonNew;          //used to read physical button status
int buttonOld = 1;      //previous button status


Scheduler     userScheduler; // to control sending of msg task
Scheduler     executeMsg // to control execution of msg task

/*---------Call back function to check status of sent data -------------*/
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {

  Serial.print("\r\nLast Packet Send Status:\t");

  //Check transmission status and display to screen [Success or Fail]
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery Fai
l");
  if (status == 0) {
    success = "Delivery Success :)";
  }
  else {
    success = "Delivery Fail :(";
  }
}
```

```
/*--------- function for receiving data -------------*/
void OnDataRecv(const uint8_t * mac_addr, const uint8_t *incomingData, int len) {
  memcpy(&newLedStatus, incomingData, sizeof(newLedStatus)); //store data to memory
  LEDState = newLedStatus.LEDstatus; //assign received data to variable
  Serial.print("I AM RECEIVING THE DATA"); //Display received data to serial monitor
  Serial.println(LEDState);

  //When data is recevied, schedule to execute msg received
  executeMsg.execute();
}


/Task scheduler function - for executing the received msg
Task executeMessage( TASK_SECOND * 0.5, TASK_FOREVER, []() {
//Execute after 500ms per function call forever
  if (LEDState == 0 ) {   //Checks remote signal sent TURN OFF
    digitalWrite(Relay, LOW); //set pin to low
    LEDnewState = 0;    //update status
    userScheduler.execute(); //send back the update
  } else if (LEDState == 1) { //TURN ON
    digitalWrite(Relay, HIGH); //set pin to high
    LEDnewState = 1;    //update status
    userScheduler.execute(); //send the update
  } else { //do nothing }

});


//Task scheduler function - send update to sensor node
Task taskSendMessage( TASK_SECOND * 0.5, TASK_FOREVER, []() {
    //Execute after 500ms per function call forever
  currentStatus.LEDstatus = LEDnewState; //update the currentStatus structure value
    /*using esp_now_send function to send the data - MAC Address, the structure mess
age variable , and the size of the msg*/
  esp_err_t result = esp_now_send(receiver, (uint8_t *)&currentStatus, sizeof(curren
tStatus)); // Send message via ESP-NOW

  //Execute call back function to check if the receiver is online
  if (result == ESP_OK) {
    Serial.println("Sent with success");  //data sent successfully
  }
  else {
    Serial.println("Error sending the data");   //sending data failed
  }
});


void setup() {

  // Initialize  Serial Monitor
  Serial.begin(115200);
  pinMode(Relay, OUTPUT); //set relay pin to as an output pin
  pinMode(switch, INPUT); //set switch pin as an input pin

  // 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;
  }
```

```
  /*---- Continuation of void setup() function ----- */


// Once ESPNow is successfully Init, we will register for Send CB to
  // get the status of Trasnmitted packet
  esp_now_register_send_cb(OnDataSent);

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

  // Add peer
  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Failed to add peer");
    return;
  }
  // Register for a callback function that will be called when data is received
  esp_now_register_recv_cb(OnDataRecv);
  userScheduler.addTask(taskSendMessage); //initialize scheduler for sending data
  taskSendMessage.enable(); //enable scheduler
   executeMsg.addTask(executeMessage); //initialize scheduler for executing rcvd msg
  executeMessage.enable(); //enable schedulure

} // end of setup()



void loop() {
  buttonNew = digitalRead(switch); //read switch status

  if ((buttonOld == 0 && buttonNew == 1)) { //Manual checking for manual switching
    if (LEDState == 0) { //check condition of LED
      digitalWrite(Relay, HIGH); //turn on LED
      LEDState = 1; //update led status
      delay(1000); //delay

    } else {
      digitalWrite(Relay, LOW);  //turn off LED
      LEDState = 0; //update LED status
      delay(1000); //delay
    }
      userScheduler.execute();  //send new update
  }
  buttonOld = buttonNew;

}  // end of void function
```

Appendix II- Sensor Node Source Code

```
/*
    Title: WSN Project - Sensor Node
    Written by: Carl Sagrado
    Student No: X00084403
    Date:       04/05/2021
    Current version:    v1.6

**Version Updates:
    v1.0 - Detect Temp and Humidity
    v1.1 - Added ESP_NOW for sending and receiving data wirelessly
    v1.2 - Added Task scheduler as an alternative solution .
            to remove unnecessary use of delay into the function
    v1.3 - Modified to execute received intructions and  to
            automatically send status to after execution
    v1.4 - Modified to communicate to the base station
    v1.5 - Modified to communicate to the end device
    v1.6 - Modified to forward rel information between base station and end device

**Desc:
    -  Communicates to the base station
    -  Communicates to the remote device
*/


#include <esp_now.h> //include ESPOW library for ESP32 to this program
#include <WiFi.h>   //include Wi-Fi library to this program
#include <TaskScheduler.h> //include task scheduler library to this program
#include <ArduinoJson.h> //include json library for arduino
#include "DHT.h"    //include DHT sensor library to this program



#define DHTPIN 22 //DHT Pin
#define DHTTYPE DHT11   // DHT 11
DHT dht(DHTPIN, DHTTYPE); //create dht object


Scheduler     userScheduler; // to control your personal task
Scheduler     userScheduler1; // to control your personal task

//  MAC Address of  receiver
uint8_t receiver[] = {0x10, 0x52, 0x1C, 0x5D, 0x62, 0x50}; //MASTER
uint8_t receiver1[] = {0xF0, 0x08, 0xD1, 0xD3, 0x3D, 0xF0}; //EXTERNAL NODE

//data Structure for msg received and sent by the base station
typedef struct struct_message {
  float temp;
  float hum;
  String device;
  int LEDstate;
} struct_message;

//data Structure for msg received and sent by end devicen
typedef struct switchStat {
  int LED;
} switchStat;


// variables to store sensor readings
float t;
float h;
String deviceN = "Room 1"; //Location of the device
```

```
// Variable to store if sending data was successful
String success;

switchStat newStatus;
switchStat currentStatus;
switchStat outgoing;
struct_message readings; //Structure example to send data (must match with the recei
veer
struct_message incomingReadings; // Create a struct_message to hold incoming sensor
readings

// 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 Fai
l");
  if (status == 0) {
    success = "Delivery Success :)";
  }
  else {
    success = "Delivery Fail :(";
  }
}
/****** RECEIVING DATA (function) *********/

// Callback when data is received
void OnDataRecv(const uint8_t * mac_addr, const uint8_t *incomingData, int len) {
  char macStr[18];
  String x;
//Store mac address to an Array
  snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x",
           mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4], mac_addr
[5]);

//add all MAC values and convert to a decimal
  x = mac_addr[0] + mac_addr[1] + mac_addr[2] + mac_addr[3] + mac_addr[4] + mac_addr
[5];

  if (x == "397") { //Detect msgs transmitted by the base station
    memcpy(&newStatus, incomingData, sizeof(newStatus));  //copy to memory
    newState = newStatus.LED; //extract rcvd data by assigning to a global variable
    Serial.println("FROM MCU -" );
    displayReceviedData();
     Serial.print("LED newState: ");
     Serial.print(newState);
     Serial.println();
     userScheduler1.execute(); //call task function to forward data to end device
  }
if (x == "969") { //F0:08:D1:D3:3D:F0 - Detect msgs sent by end device
    Serial.print("EXTERNAL NODE - ");
    memcpy(&currentStatus, incomingData, sizeof(currentStatus)); //copy  to memory
      state = currentStatus.LED; //extract rcvd data by assigning to a global var
     Serial.print("LED State: ");
     Serial.print(state);
     Serial.println();
  }
}

/****** Display received DATA (function) *********/


void displayReceviedData() { //function to display the msg received from the base st
ation
  Serial.printf("New LED status: %d \n", newState);
}
```

```
//Task scheduler function - for sending data to the base station
Task taskSendMessage( TASK_SECOND * 1, TASK_FOREVER, []() {
//Execute after 1000s per function call forever
  readings.temp = t;
  readings.hum = h;
  readings.device = deviceN;
  readings.LEDstate = state;
  esp_err_t result = esp_now_send(receiver, (uint8_t *)&readings, sizeof(readings));
 // Send message via ESP-NOW

  if (result == ESP_OK) { //check transmission status
    Serial.println("Sent with success");
  }
  else {
    Serial.println("Error sending the data");
  }
}); // start with a one second interval

//Task scheduler function - for sending data to the remove device
Task taskSendMessage1( TASK_SECOND * 0.5, TASK_FOREVER, []() {
  outgoing.LED = newState ;
  Serial.println(outgoing.LED);
  esp_err_t result = esp_now_send(receiver1, (uint8_t *)&outgoing, sizeof(outgoing))
; // Send message via ESP-NOW

  if (result == ESP_OK) { //check transmission status
    Serial.println("Sent to an end device with success");
  }
  else {
    Serial.println("Error sending the data");
  }
}); // start with a one second interval


void setup() {
  // Init Serial Monitor
  Serial.begin(115200);
  Serial.println(F("DHTxx test!"));
  dht.begin();

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

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

  // Once ESPNow is successfully Init, we will register for Send CB to
  // get the status of Trasnmitted packet
  esp_now_register_send_cb(OnDataSent);

  // Register peer
  // Add peer
  esp_now_peer_info_t peerInfo;
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  memcpy(peerInfo.peer_addr, receiver, 6);

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

```
 memcpy(peerInfo.peer_addr, receiver1, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Failed to add peer - External Node");
    return;
  }


  // Register for a callback function that will be called when data is received
  esp_now_register_recv_cb(OnDataRecv);
  userScheduler.addTask(taskSendMessage);
  userScheduler1.addTask(taskSendMessage1);
  taskSendMessage.enable();
  taskSendMessage1.enable();
}

void loop() {

  t = dht.readTemperature(); //read temperature
  h = dht.readHumidity();   //read humidity
  userScheduler.execute(); //call task scheduler to send update

}
```

Appendix III- Base Station Source Code

```
/*
    Title: WSN Project - Base Station
    Written by: Carl Sagrado
    Student No: X00084403
    Date:       04/05/2021
    Current version:    v1.3

**Version Updates:
    v1.0 - Receive data from Sensor 1 and Sensor 2
    v1.2 - Convert sensor data to a json format
    v1.3 - Detect and read data messages

**Desc:
    - The base station acts as a sink node of the WSN. This node receives all
        the sensor data and forward to the web server.
    - Base station transmits data received from server to designated sensor node
*/


/*Import libraries */
#include <esp_now.h> //include ESPOW library for ESP32 to this program
#include <WiFi.h>   //include Wi-Fi library to this program
#include <TaskScheduler.h> //include task scheduler library to this program
#include <ArduinoJson.h> //include json library for arduino

Scheduler     userScheduler; // to control task tas
Scheduler     userScheduler1; // to control another task


//structure for serial data
typedef struct switchStat {
  int LED;
} switchStat;

//structure for wireless data
typedef struct struct_message {
  float temp; //temperature
  float hum; //humidity
  String device; // device name
  int LEDstate; //remote device state
} struct_message;

//Receiver MAC address
uint8_t receiver[] = {0x10, 0x52, 0x1C, 0x62, 0xB4, 0x08}; //Sensor Node 1
uint8_t receiver1[] = {0xF0, 0x08, 0xD1, 0xD2, 0x83, 0xF0}; //Sensor Node 2

// Define variables to store incoming readings
float incomingTemp;
float incomingHum;
String deviceName;
int ledStateR;

// Variable to store if sending data was successful
String success;
switchStat nodeRD1; //structure to send to node 1
switchStat nodeRD2; //structure to send to node 2

// Create a struct_message to hold incoming sensor readings
struct_message incomingReadings;

//Task scheduler function - for sending data to Sensor Node 1
Task taskSendMessage( TASK_SECOND * 0.5, TASK_FOREVER, []() {
  esp_err_t result = esp_now_send(receiver, (uint8_t *)&nodeRD1, sizeof(nodeRD1)); /
/ Send message via ESP-NOW
}); //Execute after 500ms per function call forever
```

```
//Task scheduler function - for sending data to Sensor Node 2
Task taskSendMessage1( TASK_SECOND * 0.5, TASK_FOREVER, []() {
  esp_err_t result = esp_now_send(receiver1, (uint8_t *)&nodeRD2, sizeof(nodeRD2));
// Send message via ESP-NOW
});//Execute after 500ms per function call forever

// Callback when data is received
void OnDataRecv(const uint8_t * mac_addr, const uint8_t *incomingData, int len) {

  memcpy(&incomingReadings, incomingData, sizeof(incomingReadings));
  incomingTemp = incomingReadings.temp; //readings for temperature
  incomingHum = incomingReadings.hum; //readings for humidity
  deviceName = incomingReadings.device; //device name
  ledStateR = incomingReadings.LEDstate; //remote device status
  receviedData(); //function call for
}

//function to convert received data to JSON format to send to server
void receviedData() {

  String msg2serial;
  DynamicJsonDocument message(1024);
  //Serial.printf("Received: LED:%s Temp: %sc Hum: %s\n", LED, Temp, Hum);
  message["LEDState"] = ledStateR;
  message["TEMP"] = incomingTemp ;
  message["HUM"] = incomingHum;
  message["deviceName"] = deviceName;
  serializeJson(message, msg2serial);
  Serial.println(msg2serial); //Send to Server via serial
}

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

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

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


  // Once ESPNow is successfully Init, we will register for Send CB to
  // get the status of Trasnmitted packet
  esp_now_register_send_cb(OnDataSent);


   // Register peer
  // Add peer
  esp_now_peer_info_t peerInfo;
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  memcpy(peerInfo.peer_addr, receiver, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Failed to add peer - MCU");
    return;
  }
  memcpy(peerInfo.peer_addr, receiver1, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Failed to add peer - External Node");
    return;
  }
```

```
 // Register for a callback function that will be called when data is received
  esp_now_register_recv_cb(OnDataRecv);
  userScheduler.addTask(taskSendMessage);
  taskSendMessage.enable();
  userScheduler1.addTask(taskSendMessage1);
  taskSendMessage1.enable();
}
/*End of setup() */


void loop() {

  if (Serial.available()) {
    String command = Serial.readStringUntil('\n');
    // Serial.println(command);

    if (command.equals("R1ON")) {
      nodeRD1.LED = 1;
      userScheduler.execute();
      // Serial.println(R1_LED.LED);
    } else if (command.equals("R1OFF")) {
      nodeRD1.LED = 0;
      userScheduler.execute();
    } else if (command.equals("R2ON")) {
      nodeRD2.LED = 1;
      //Serial.println(R1_LED.LED);
      userScheduler1.execute();
      // Serial.println(R1_LED.LED);
    } else if (command.equals("R2OFF")) {
      nodeRD2.LED = 0;
      userScheduler1.execute();

    }    else if (command.equals("R2OFFR1OFF") || command.equals("R1OFFR2OFF")) {
      nodeRD1.LED = 0;
      nodeRD2.LED = 0;
      userScheduler.execute();
      userScheduler1.execute();

    } else if (command.equals("R2ONR1ON") || command.equals("R1ONR2ON")) {
      nodeRD1.LED = 1;
      nodeRD2.LED = 1;
      userScheduler.execute();
      userScheduler1.execute();

    } else {      }
  }
}
/*End of void () */
```

Appendix IV – Web Server Source code

```
/*
    Title: WSN Project - Web Server
    Written by: Carl Sagrado
    Student No: X00084403
    Date:       04/05/2021
    Current version:    v1.3

**Version Updates:
    v1.0 - Express and Socket.io to serve HTTP request
    v1.1 - Read sensor data via serial using serialport module
    v1.3 - Emit received data to the webpage

**Desc:
    - Web server to serve http request and communicates to the wireless sensor netwo
rk
*/



var express = require("express"); //require express app
//Express app setup
var app = express(); //invoke express function
var http = require('http').Server(app);
app.use(express.static('public')); //express app to server the public folder
var listenFunction = http.listen(4000, function () { //listen requests on port 4000
    console.log('listening on *:4000');
});


//Setup socket.io
var io = require('socket.io')(http);
var R1deviceStat; //status of the R1 device
var R2deviceStat; //status of the R2 device
var listenCount;

var SerialPort = require('serialport'); //import SerialPort module
var Readline = require('@serialport/parser-readline');


var connect2Serial = function () {
    /*Creating a new instance of Serial port that takes in parameter values such as
commport value, baudrate, and error checking function*/
    var port = new SerialPort('COM9', { baudRate: 115200 }, function (err) {
        if (err) {
            return console.log('Error on port open: ', err.message);
        }
        console.log('Serial Port Open');
    });

    var parser = port.pipe(new Readline());  // seperate data input using newlines


    /*************Socket.io connection*****************/
    var activeClients = 0;  //number of connected clients
    var state = 0;  //led state 0 off, 1 on

    io.on('connection', function (socket) {
        socket.on('disconnect', function () {
            activeClients -= 1;
            io.emit('nClientUpdate', {clients: activeClients, clientLogger: state});
        });
```

```
 activeClients += 1;
    io.emit('nClientUpdate', { clients: activeClients, clientLogger: state });

    socket.on('r1Sw', function (data) { //Sending R1 switch status on change
        //  console.log(data.r1VAL);
        port.write(data.r1VAL, function (err) {
            if (err) {
                return console.log('Error on port write: ', err.message);
            }
            console.log('message written ' + data.r1VAL);
        });
    });

    socket.on('r2Sw', function (data) { //Sending R1 switch status on change
        // console.log(data.r2VAL);
        port.write(data.r2VAL, function (err) {
            if (err) {
                return console.log('Error on port write: ', err.message);
            }
            // console.log('message written ' + data.r1VAL );
        });
    });


});


  activeClients += 1;
    io.emit('nClientUpdate', { clients: activeClients, clientLogger: state });

    socket.on('r1Sw', function (data) { //Read R1 switch status on change
        //  console.log(data.r1VAL);
        port.write(data.r1VAL, function (err) { //write to port

            if (err) {
                return console.log('Error on port write: ', err.message);
            }
            console.log('message written ' + data.r1VAL);
        });
    });

    socket.on('r2Sw', function (data) { //Read R2 switch status on change
        port.write(data.r2VAL, function (err) { //write to port
            if (err) {
                return console.log('Error on port write: ', err.message);
            }
        });
    });
});
parser.on('data', function (data) {
    console.log(data);

    var isValidJSON;
    /*String values that can cause server to crash*/
    var start1 = data.includes("ets");
    var start2 = data.includes("rst:0x1");
    var start3 = data.includes("waiting");
    var start4 = data.includes("Start");
    var start5 = data.includes("Changed");
    var start6 = data.includes("dhcps:");
    var start7 = data.includes("Sent");
    var start8 = data.includes("R1ON");
    var start9 = data.includes("R1OFF");
    var start10 = data.includes("R2ON");
    var start11 = data.includes("R2OFF");
    var start12 = data.includes("msgSenttoR2");
    var start13 = data.includes("Not here");
```

```
 /*Detect String Values*/
        if (start2 || start3 || start4 || start5 || start6) {
          console.log(start1 + " " + start2 + " " + start3 + " " + start4 + " " + st
art5 + " " + start6);
          isValidJSON = false;
          setTimeout(function () { console.log("I AM HERE"); }, 5000);
        } else if (start1) {
          console.log(start1 + "ETS");
          isValidJSON = false;
          setTimeout(function () { console.log("I AM HERE"); }, 10000);
        } else if (start7 || start8 || start9 || start10 ||start11) {
         // console.log(data);

          console.log(data);
        }   else if (start12 || start13){
          console.log(data);

        } else {
          isValidJSON = true;
        }

        if (isValidJSON == true) { //if no error has occurred
          var test = JSON.parse(data); // re-parsing

          if (test.deviceName == "Room 2") { //"Room 2 settings"
            var r2temp = test.TEMP; //strore temp value
            var r2hum = test.HUM; //strore humidity value
            var r2LEDstate = test.LEDState; //strore remote device status
            var r2device = test.deviceName; //strore device name
            R2deviceStat = "Connected"; //device status

            if (r2temp != "null") {
              io.emit('R2TEMPState', { R2temp: r2temp }); //emit to the page
            }

            if (r2hum != "null") {
              io.emit('R2HUMState', { R2humidity: r2hum }); //emit to the page
            }
            if (test.R2deviceName != "null") {
              io.emit('R2ActiveDeviceState', { R2active: r2device }); //emit stat
              io.emit('R2DeviceState', { R2DeviceStat: R2deviceStat });//emit
            }

            if (r2LEDstate != "null") {
              io.emit('R2LEDState', { R2LED: r2LEDstate }); //emit to page
              r1activeDevice = + 1;
              r1deviceStat = "Connected";

              io.emit('DeviceCount', { DeviceCount: r1activeDevice });
            }
          }

            if (test.deviceName == "Room 1") { //"Room 1 settings"
                     var r1temp = test.TEMP;
                     var r1hum = test.HUM;
                     var r1LEDstate = test.LEDState;
                     var r1device = test.deviceName;
                     var r1activeDevice;
                     R1deviceStat = "Connected";

                     io.emit('R1ActiveDeviceState', { R1active: r1device});
                     if (r1LEDstate != "null") {
                       io.emit('R1LEDState', { R1LED: r1LEDstate }); //emit RDstatus
                       r1activeDevice = + 1;
                       r1deviceStat = "Connected";
                       io.emit('DeviceCount', { DeviceCount: r1activeDevice });
                     }
```

```
            if (r1temp != "null") {
                        io.emit('R1TEMPState', { R1temp: r1temp });
                        io.emit('R1DeviceState', { R1DeviceStat: R1deviceStat });
                      }
                      if (r1hum != "null") {
                        io.emit('R1HUMState', { R1humidity: r1hum });
                      }

                    }
                  }


                });

          port.on('close', function () { // function to listen if port is closed
              console.log('Port closed');
              reconnectArd(); // function call to reconnect
              R1deviceStat = "Disconnected";
          })

          port.on('error', function (err) {  //listen for any connection errors
              console.log('Error on port: ', err.message);
              reconnectArd(); // function call to reconnect
          });

          if (listenCount == 0) {
              listenFunction();
              listenCount++;
          }

  }

   /*************** END OF Connect2 Serial function ************/



connect2Serial(); //Call function
console.log("PASSED through here") // display if connection lost

var reconnectArd = function () { // Function to reconnect when serial port is closed
    R1deviceStat = "Disconnected";
    console.log('INITIATING RECONNECT');
    setTimeout(function () { // wait for 4s and then attempt to reconnect
        console.log('RECONNECTING TO ARDUINO');
        connect2Serial();
    }, 4000);
};
```

Appendix V – Webpage Source Code (HTML)

```
<!DOCTYPE html>
<!-- Prototype -->
<html>

<head>
    <link href="css/font-face.css" rel="stylesheet" media="all">
    <script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.5.0/Chart.min.js"></sc
ript>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.2/css/all.css"
        integrity="sha384-
fnmOCqbTlWIlj8LyTjo7mOUStjsKC4pOpQbqyi7RrhN7udi9RwhKkMHpvLbHG9Sr" crossorigin="anonymous"
>
    <link rel="stylesheet" href="css/style.css" />
    <script src="/socket.io/socket.io.js"></script>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></scri
pt>
    <script src="script/script.js"></script>


</head> <!-- END OF html head tag -->

<body id="body"> <!-- Start of body tag -->
    <div class="container">
            <!-- First ROW -->
        <div class="first_row">
            <!-- Project TITLE STARTS HERE -->
            <div class="project_title">
                <div>
                    <div> <!-- Project Name -->
                        <div class="project__greeting">
                            <img src="assets/LOGOicon.svg" alt="" />
                            <h1>IOT Project - Prototype</h1>
                            <p>Welcome to your IOT dashboard</p>
                        </div>
                    </div>
                </div>
                <!-- CLOCK -->
                <div class="clock" style="float:right !important">
                    <span id="datetime"></span>
                </div>
            </div>
        </div> <!-- Project TITLE ENDS HERE -->

        <div class="second__row">
            <!-- Second ROW ROW -->
            <!-- Room 1- FIRST COLUMN-->
            <div class="card-main">
                <div class="card-top">
                    <!--LOCATION & STATUS-->
                    <p>Location: <span class="nodeTitle" id="NodeName">--</span> </p>
                    <p>Status: <span class="nodeMargin" id="NodeStat">offline</span> </p>
                </div>
                <div>
                    <!--TEMP  -->
                    <div class="card_inner_1_top">
                        <!--TEMP ICON -->
                        <i class="fas fa-thermometer-half fa-
                             2x" style="color:#ee2917b4;" aria-hidden="true"></i>
                        <span class="card-val">
                            <span id="R1TEMP_val"> 0 </span>
                            <!--TEMP VALUE HERE -->
                            <sup>&deg;C</sup>
                        </span>
                    </div>
                    <div>
                        <p class="text-primary-p">Temperature</p>
                    </div>
                </div>
```

```
                <div class="card_inner_2">
                    <!--HUMIDITY -->
                    <div class="card_inner_2_top">
                        <!--HUM ICON -->
                        <i class="fas fa-tint fa-2x" style="color:#2196f5;" aria-
                            hidden="true"></i>
                        <span class="card-val">
                            <span id="R1HUM_val">0</span>
                            <!--HUMIDITY VALUE HERE -->
                            <sup class="units2">&percnt;</sup>
                        </span>
                    </div>
                    <div>
                        <p class="text-primary-p">Humidity</p>
                    </div>
                </div>
                <div>
                    <p>Remote Devices: <span id="NodeCount">0</span> </p>
                    <!--Connected Devices HERE -->
                </div>
            </div> <!-- END OF Room 1 -->

            <!-- Room 2 - SECOND COLUMN-->
            <div class="card-main">
                <div class="card-top">
                    <!--LOCATION & STATUS-->
                    <p>Location: <span class="nodeTitle" id="R2NodeName">--</span> </p>
                    <p>Status: <span class="nodeMargin" id="R2NodeStat">offline</span> </
                        p>
                </div>

                <div>
                    <div class="card_inner_1_top">
                        <!--TEMP  -->
                        <i class="fas fa-thermometer-half fa-
                            2x" style="color:#ee2917b4;" aria-hidden="true"></i>
                        <!--ICON -->
                        <span class="card-val">
                            <span id="R2TEMP_val"> 0 </span>
                            <!--TEMP  VALUE HERE-->
                            <sup>&deg;C</sup>
                        </span>
                    </div>
                    <div>
                        <p class="text-primary-p">Temperature</p>
                    </div>
                </div>

                <div class="card_inner_2">
                    <div class="card_inner_2_top">
                        <!--HUMIDITY  -->
                        <i class="fas fa-tint fa-2x" style="color:#2196f5;" aria-
                        hidden="true"></i>
                        <!--HUMIDITY ICON  -->
                        <span class="card-val">
                            <span id="R2HUM_val">0</span>
                            <!--HUMIDITYVALUE HERE  -->
                            <sup class="units2">&percnt;</sup>
                        </span>
                    </div>
                    <div>
                        <p class="text-primary-p">Humidity</p>
                    </div>
                </div>
                <div>
                    <p>Remote Devices: <span id="NodeCount">0</span> </p>
                    <!--Connected Devices HERE -->
                </div>
            </div> <!-- END OF Room 2 -->
```

```
<!-- Room 3 - THIRD COLUMN-->
<div class="card-main">
    <div class="card-top">
        <!--LOCATION & STATUS-->
        <p>Location: <span class="nodeTitle" id="NodeName">--</span> </p>
        <p>Status: <span class="nodeMargin" id="NodeStat">offline</span> </p>
    </div>
    <div>
        <!--TEMP -->
        <div class="card_inner_1_top">
            <!--TEMP ICON   -->
            <i class="fas fa-thermometer-half fa-
            2x" style="color:#ee2917b4;" aria-hidden="true"></i>
            <span class="card-val">
                <span id="R1TEMP_val"> 0 </span>
                <!--TEMP VALUE HERE -->
                <sup>&deg;C</sup>
            </span>
        </div>
        <div>
            <p class="text-primary-p">Temperature</p>
        </div>
    </div>

    <div class="card_inner_2">
        <!--HUMIDITY -->
        <div class="card_inner_2_top">
            <i class="fas fa-tint fa-2x" style="color:#2196f5;" aria-
            hidden="true"></i>
            <!--HUM ICON -->
            <span class="card-val">
                <span id="R1HUM_val">0</span>
                <!--HUMIDITY VALUE HERE -->
                <sup class="units2">&percnt;</sup>
            </span>
        </div>
        <div>
            <p class="text-primary-p">Humidity</p>
        </div>
    </div>
    <div>
        <p>Remote Devices: <span id="NodeCount">0</span> </p>
        <!--REMOTE DEVICES COUNT HERE -->
    </div>
</div>


<!-- Light switches  -->
<!-- Title -->
<div class="card-2-row2">
    <div style="text-align: center; font-size: 12px; font-style:bold;">
        <p> Light Switches </p>  <!-- TITLE -->
    </div>
    <!-- Room 1 SWITCH -->
    <div class="card-control_1">
        <div class="card_inner_3_top">
            <p class="roomName" style="font-size: 13px;">Room 1:</p>
            <span class="remote-link" id="device1_stat">offline</span>  <!-
            - switch connectivity status -->
            <span class="card-val">
                <p> OFF </p>
                <label class="switch"> <!-- ROOM 1 SWTICH gui-->
                    <input type="checkbox" id="room_1_sw">
                    <span class="slider round"></span>
                </label>
                <p>ON</p>
            </span>
        </div>
    </div>  <!-- END OF Room 1 SWITCH -->
```

```
            <!-- Room 2 SWITCH -->
              <div class="card-control_1">
                  <div class="card_inner_3_top">
                      <p class="roomName" style="font-size: 13px;">Room 2:</p>
                      <span class="remote-link" id="device2_stat">offline</span> <!
                    - switch connectivity status -->
                      <span class="card-val">
                          <p> OFF </p>
                          <label class="switch"> <!-- ROOM 2 SWTICH gui-->
                              <input type="checkbox" id="room_2_sw">
                              <span class="slider round"></span>
                          </label>
                          <p>ON</p>
                      </span>
                  </div>
              </div> <!-- END OF Room 2 SWITCH -->

              <!-- Room 3 SWITCH -->
              <div class="card-control_1">
                  <div class="card_inner_3_top">
                      <p class="roomName" style="font-size: 10px;">Living Room:</p>
                      <span class="remote-link-1" id="device1_stat">offline</span> <!
                      - switch connectivity status -->
                      <span class="card-val">
                          <p> OFF </p>
                          <label class="switch"> <!-- ROOM 3 SWTICH gui-->
                              <input type="checkbox">
                              <span class="slider round"></span>
                          </label>
                          <p>ON</p>
                      </span>
                  </div>
              </div> <!-- END OF Room 3 SWITCH -->
          </div>  <!-- END OF Light switches column  -->
      </div> <!-- END OF main container (4 columns)  -->


      <div class="third_row">
          <div style="height: 100px; width: 600px;">
              <canvas id="Chart"></canvas>
          </div>
      </div>

</body> <!-- END OF body   -->

</html> <!-- ENDOF HTML   -->
```

Appendix VI – Webpage Source Code (JavaScript)

```
$(document).ready(function () { /*When document is ready*/

    /*------- Display local time  */
    var dt = new Date();
    document.getElementById("datetime").innerHTML = (("0" + dt.getHours()).slice(-
2)) + ":" + (("0" + dt.getMinutes()).slice(-2));

    /*------- Display Chart -------- */
    var currentInterval = 0;
    var ctx = document.getElementById('Chart').getContext('2d');

    var chartConfig = {
        type: 'line',
        data: {
            labels: [],
            datasets: [{
                fill: false,
                label: 'Sensor Data',
                backgroundColor: 'blue',
                borderColor: 'blue',
                data: []
            }]
        },
    };

    var chart = new Chart(ctx, chartConfig);

    var R1Connection = "Disconnected"; //enable and disable the switch in R1

    var socket = io();      // using socket.io
    socket.on('nClientUpdate', function (msg) {
        // $("#client_count").html(msg.clients);
        $("#led_val").html(msg.clientLogger);
    });

    /*Sensor data of node 1*/

    socket.on('R1TEMPState', function (msg) {   //temperature

        $("#R1TEMP_val").html(msg.R1temp);

        chartConfig.data.labels.push(currentInterval + "s");

        chartConfig.data.datasets.forEach(function (dataset) {
            dataset.data.push(msg.R1temp);
        });

        if (currentInterval > 60) {     /*Chart to display every */
            chart.destroy();
            currentInterval = 0;

            chartConfig.data.labels = [];
            chart = new Chart(ctx, chartConfig);

            chartConfig.data.labels.push(currentInterval + "s");

            chartConfig.data.datasets.forEach(function (dataset) {
                dataset.data.push(msg.R1temp);
            });
        };
        currentInterval += 1;
        chart.update();
    });

    socket.on('R1HUMState', function (msg) { //listen for humidity data event
        $("#R1HUM_val").html(msg.R1humidity); // apply humidity value to html
    });
```

```
socket.on('R1ActiveDeviceState', function (msg) {// active device
        $("#NodeName").html(msg.R1active); // apply data status to html
    });

    socket.on('DeviceCount', function (msg) { //device count
        $("#NodeCount").html(msg.DeviceCount); // apply count to html
    });

    socket.on('R1DeviceState', function (msg) { //device status
        $("#NodeStat").html(msg.R1DeviceStat);
        if (msg.R1DeviceStat == "Disconnected") {
            $("#device1_stat").html("Disconnected");
            R1Connection = "Disconnected";
        } else {
            $("#device1_stat").html("Connected");
            R1Connection = "Connected";
        }
    });

    /*Sensor data of node 2*/
    socket.on('R2ActiveDeviceState', function (msg) { // active device
        $("#R2NodeName").html(msg.R2active); // apply value to html
    });

    socket.on('R2HUMState', function (msg) { //listen for humidity data event
        $("#R2HUM_val").html(msg.R2humidity);
    });

    socket.on('R2TEMPState', function (msg) { //temperature
        $("#R2TEMP_val").html(msg.R2temp);
    });

    socket.on('R2DeviceState', function (msg) {  //device status
        $("#R2NodeStat").html(msg.R2DeviceStat);

        if (msg.R2DeviceStat == "Disconnected") {
            $("#device2_stat").html("Disconnected");
            R2Connection = "Disconnected";

        } else {
            $("#device2_stat").html("Connected");
            R2Connection = "Connected";
        }
    });

   /*------Switch button interaction -----
    var r1swVal;
    var r2swVal;
    var r1button;
    var r2button;
    var loadChecker = 1;
    var loadChecker2 = 1;

    var loadConditionR1 = document.getElementById('room_1_sw');
    var loadConditionR2 = document.getElementById('room_2_sw');
    var switchR1 = document.querySelector('input[id="room_1_sw"]');
    var switchR2 = document.querySelector('input[id="room_2_sw"]');

    /*When page is loaded, check position of the switch */

    /*Check end device status of connected to sensor node 1  and apply */
    socket.on('R1LEDState', function (msg) {
        r1button = msg.R1LED;
        if (loadChecker == 1) {
            if (r1button == 1) {
                loadConditionR1.checked = true;
            } else {
                loadConditionR1.checked = false;
            }
        }
          loadChecker++;
    });
```

```
 /*Check end device status of connected to sensor node 2  and apply */
    socket.on('R2LEDState', function (msg) {
        r2button = msg.R2LED;
        if (loadChecker2 == 1) {
            if (r2button == 1) {
                loadConditionR2.checked = true;
            } else {
                loadConditionR2.checked = false;
            }
        }
        loadChecker2++;
    });



    /*---When user toggle the switch ---*/

     /*Listen to switch change of Room 1*/
    switchR1.addEventListener('change', function () {

        if (R1Connection == "Connected") {
            if (switchR1.checked) {

                r1swVal = "R1ON\n";
                socket.emit('r1Sw', { r1VAL: r1swVal });  //send R1 switch value to hte s
erver
            } else {
                r1swVal = "R1OFF\n";
                socket.emit('r1Sw', { r1VAL: r1swVal });  //send R1 switch value to hte s
erver
            }

        }
        console.log(r1swVal);
    });

        /*Listen to switch change of Room 2*/
    switchR2.addEventListener('change', function () {
        if (R2Connection == "Connected") {
            if (switchR2.checked) {
                r2swVal = "R2ON\n";
                socket.emit('r2Sw', { r2VAL: r2swVal });  //send R1 switch value to hte s
erver
                switchR2.checked = true;
            } else {
                r2swVal = "R2OFF\n";
                socket.emit('r2Sw', { r2VAL: r2swVal });  //send R1 switch value to hte s
erver
                switchR2.checked = false;
            }
        }
        console.log(r2swVal);
    });

       /*Listen to switch change of Room 3*/
    switchR3.addEventListener('change', function () {
        if (R3Connection == "Connected") {
            if (switchR3.checked) {
                r2swVal = "R3ON\n";
                socket.emit('r3Sw', { r2VAL: r3swVal });  //send R1 switch value to hte s
erver
                switchR3.checked = true;
            } else {
                r3swVal = "R3OFF\n";
                socket.emit('r3Sw', { r3VAL: r3swVal });  //send R1 switch value to hte s
erver
                switchR3.checked = false;
            }
        }
        console.log(r3swVal);
    });
});
```

Appendix VII – Webpage Source Code (CSS)

```
      /* --------- IMPORT FONTS -----------*/
@import url("https://fonts.googleapis.com/css2?family=Lato:wght@300;400;700&display=swap");
@import url('https://fonts.googleapis.com/css2?family=Poppins:wght@200&display=swap');


       /* ----- default setup for body tag  ------ */
      body {
          box-sizing: border-box;
          font-family: "Poppins", sans-serif;
          background-color: rgba(99, 124, 124, 0.158);
      }

      .container {    /* - page setup template -- */
          grid-template-rows: 0.2fr 3fr;
      }

      /*---first row container--- */
      .first_row {
          padding: 20px 35px ;
      }

      /* Project TITLE CSS  */
      .project_title {
          display: flex;
          align-items: center;
          justify-content: space-between;
      }

      .project__greeting{
          width: 500px;
      }
      .project__greeting img {
          max-height: 75px;
          object-fit: contain;
          margin-right: 20px;
          justify-content: space-around;
          float: left;

      }

      .project__greeting>h1 {
          font-size: 24px;
          color: teal;
        margin : 0;
      }

      .project__greeting>p {
          font-size: 14px;
          color: #777c80;
          margin:0;
      }

      .clock {
          display: flex;
          align-items: right;
          justify-content: flex-end;
          font-size: 70px;
          margin-right: 70px;
          font-weight: bold;
      }
      /* Project TITLE CSS - ENDS HERE */

      /*--SENSOR DATA Chart  --*/
      .chart {
          width: 20%;
          height: 80%;
      }



      /*Second row template*/
```

```
        /*Second row template*/

        .text-primary-p {
            color: #7f8486;
            font-size: 12px;
        }

        .second__row {
            padding-left: 30px;
            display: grid;
            grid-template-columns: 1fr 1fr 1fr 3fr;
            gap: 30px;
            margin: 20px 0;
        }

        /*third row template*/
        .third_row {

            padding-left: 30px;
            gap: 30px;
            width: 280px;
            margin-top: 20px;
            padding-bottom: 20px;
            display: flex;
            justify-content: space-between;
        }

        /* Cards Edits-- */


        .card_inner_1_top {
            margin-left: 12px;
            display: flex;
            justify-content: space-between;
        }


      /*Light switches column*/
        .card_inner_1_top p {
            padding: 0px;
            margin:0px
        }

        .card_inner_2 {
            padding-top: 0px;
        }

        .card_inner_2_top {
            margin-left: 10px;
            display: flex;
            width: 100%;
            justify-content: space-between;

        }

        .card_inner_2_top>p {
            font-size: 15px;
            margin-left: 8px;
            display: flex;
            justify-content: space-between;
        }


        /*Light switches column*/
        .card_inner_3_top>p {
            font-size: 15px;
            margin-left: 8px;
            display: flex;
            justify-content: space-between;
        }
```

```
.card_inner_3_top {
    margin-left: 10px;
    display: flex;
    width: 100%;
    justify-content: space-between;

}

.roomName{
    margin-top: 0px;
    margin-bottom: 0px;
}


.card_inner_3_top p {
    padding: 0px;
}

.remote-link {
    display: flex;
    align-items: center;
    font-size: 10px;
    font-weight: 200px;
    color: green;
     margin-left: 13px;

}

.remote-link-1 {
    display: flex;
    align-items: center;
    font-size: 10px;
    font-weight: 200px;
    color: green;
     margin-left: 5px;

}

.nodeTitle{
    margin-left:15px;
}

.nodeMargin{
    margin-left: 27px;
}


/* Three column cards -  */

.card-main {
    display: flex;
    flex-direction: column;
    justify-content: space-around;
    height: 280px;
    padding: 25px;
    border-radius: 5px;
    background: #ffffff;
    box-shadow: 5px 5px 13px #ededed, -5px -5px 13px #ffffff;
    width: 250px;
    padding-top: 20px;
}

.card-main>span {
    font-size: 15px;
}

.card-2-row2 {
    display: flex;
    flex-direction: column;
    justify-content: space-around;
    height: 150px;
    border-radius: 5px;
    background: #ffffff;
    box-shadow: 5px 5px 13px #ededed, -5px -5px 13px #ffffff;
    width: 250px;
}
```

```
.card-2-row2>span {
    font-size: 15px;
}

.card-val {
    color: #454747;
    justify-content: space-between;
    display: flex;
    /* padding-top: 10px; */
    margin-left: 20px;
    /* padding-right: 10px; */
}

.card-val>sup {
    /*wrapping the output value + unit value*/

    font-size: 15px;
    margin-left: 0px;
}

.units2 {
    margin-right: 10px
}


.card-control_1 {
    display: flex;
}

.card-val>span {
    font-size: 35px;
    margin-left: 0px;

}

.card-val>p {
    margin-left: 5px;
    margin-right: 5px;
    font-size: 10px;
}


.card-top{
    padding-top: 30px;
}

.card-top p {

    color: #1e1f20ee;
    font-size: 11px;

}

.card-top>p>span {

    color: #1e1f20ee;
    /* margin-left: 20px; */
    font-size: 15px;

}


/*SWITCH BUTTON - https://www.w3schools.com/howto/howto_css_switch.asp */
.switch {
    position: relative;
    display: inline-block;
    width: 38px;
    height: 22px;
}

.switch input {
    opacity: 0;
    width: 0;
    height: 0;
}
```

```
.slider {
    position: absolute;
    cursor: pointer;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    background-color: #ccc;
    -webkit-transition: .4s;
    transition: .4s;
}

.slider:before {
    position: absolute;
    content: "";
    height: 18px;
    width: 18px;
    left: 2px;
    bottom: 2px;
    background-color: white;
    -webkit-transition: .4s;
    transition: .4s;
}

input:checked+.slider {
    background-color: rgba(0, 117, 0, 0.596);
}

input:focus+.slider {
    box-shadow: 0 0 1px rgba(0, 117, 0, 0.596);
}

input:checked+.slider:before {
    -webkit-transform: translateX(16px);
    -ms-transform: translateX(16px);
    transform: translateX(16px);
}

/* Rounded sliders */
.slider.round {
    border-radius: 50px;
}

.slider.round:before {
    border-radius: 50%;
}
```