Microcontroller-based Consumer Level Weather Prediction System

by M. G. Akila Kavinda

198741T

Faulty of Information Technology

University of Moratuwa

July 2022

Microcontroller-based Consumer Level Weather Prediction System

M. G. Akila Kavinda

198741T

Dissertation submitted to the Faculty of Information Technology, University of Moratuwa, Sri Lanka for the partial fulfilment of the requirements of Degree of Master of Science in Information Technology

July 2022

Declaration

I declare that this thesis is my work and has not been submitted in any form for another degree or diploma at any university or other institution of tertiary education. Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.

Name of Student: Signature of Student

M. G. Akila Kavinda ……

Date:

Supervised by:

Name of Supervisor Signature of Supervisor

Mr B. H. Sudantha ……..……

Date:

Dedication

I dedicate this research thesis lovingly to the followings.

Mr B. H. Sudantha, my research supervisor for dedicating his valuable time with me even though he is busy with his workload and the guidance provide me to follow the correct path.

Dr S. C. Premaratne, course coordinator, Dr C. P. Wijesiriwardena, Research Coordinator and Dr (Ms.) Kulani Mahadewa, the research supervisor for guiding me to the correct path when searching for a suitable research topic due to my non-IT background.

My mother for providing me with continuous support and encouragement throughout the degree programme.

Thank you.

Acknowledgements

First, I would like to express my sincere gratitude to Mr B. H. Sudantha who is my research supervisor and the Dean of the Faculty of Information Technology, the University of Moratuwa for providing me with the correct guidance and suitable methods throughout my research study even though allocate a time of his busy work schedule.

Then, I would like to express my sincere gratitude to Prof. Mohamad Ferdihous who taught Research Methodology and Literature Review and Thesis Writing subjects which were greatly helpful for this research study.

Then, I would like to express my sincere gratitude to my mother for providing me with support and encouragement throughout this degree programme.

Then, I would like to express my sincere gratitude to Mr Danuka Dilan Perera who is a director of Digital Electronics (Pvt) Ltd. for sponsoring the required components, development boards and all hardware parts required to develop this weather prediction system. Without his support, this development would become a nightmare for me.

Finally, I would like to express my sincere gratitude to Mr Chamodh Hettiarachchige for providing the support for syncing the data to an IoT cloud service.

Abstract

Weather conditions are important to people in planning their activities. Since weather conditions changed unexpectedly, monitoring weather conditions are very important these days. Weather stations are built to fulfil that purpose. But, if your area is not covered by that kind of weather station, you will not get accurate weather forecasts. Therefore, it can be caused to facing unexpected weather conditions. A system proposed in this paper will collect weather-related data and a live preview of those data can be seen on a display. All the outputs will display in a Nextion Display, which provides a smooth and fast response due to its internal microcontroller. Then those captured data will be logged in to a remote server. Also, this outdoor unit is powered by solar energy as a sustainable energy source. From this unit, temperature, humidity, atmospheric pressure, precipitation, wind speed and direction will be measured. Upcoming weather can be determined using the changes in atmospheric pressure. Based on those data, by using the zambretti algorithm, the upcoming weather condition will be predicted. This weather prediction system will be developed with microcontrollers which come affordable yet more powerful to handle this kind of system. Therefore, weather data will be captured by using Arduino Mega 2560 microcontroller board. Then it will be transmitted wirelessly to the indoor unit that handles the processing of those data, displays predicted data and the data logging to an Adafruit IO Internet of Things (IoT) cloud service. Those tasks will be handled by the ESP32 microcontroller board.

Keywords: Arduino, ESP32, Weather Prediction, Data Logging, Weather Station

Table of Contents

List of Figures

List of Tables

Chapter 1 - Introduction

1.1 Introduction to the study

Weather can be described as the state of the atmosphere in a particular place for a short period [1] [2]. It involves temperature, humidity, atmospheric pressure, precipitation, wind speed etc. A weather forecast is a prediction of the behaviour of weather in a certain period [3] [4]. Weather forecast enables people to plan their day, farmers to adjust their activities, travellers to plan their trips etc. Therefore, if we can obtain how the upcoming weather will be, that would be much easier for our life [5].

If you can capture weather-related data such as temperature, humidity, air pressure, precipitation, wind speed and direction etc. in your location and if you can predict the upcoming weather prediction, that would ease your lifestyle. Therefore, considering the global level, there are professional-grade weather stations are available to capture weather-related data. Those units have an outdoor unit to capture the weather-related data and it will be using wireless or wired communication medium to transfer those data to the indoor unit where the consumer can view the captured data. Most of those units are quite expensive to purchase and only high-end devices have more data analytics parts and IoT-based dashboards. Davis Instruments, Ambient Weather, La Crosse Technology, Aercus Instruments, Netatmo, Tesa and AcuRite can be considered industry-leading consumer-level weather station manufacturers. Using their weather stations, the above-mentioned data can be captured and some high-end models have data backup facilities for their IoT cloud services.

Since weather stations are capable to capture weather-related data, if is there any possibility to predict the upcoming weather, it would be an added advantage to the consumers. Some weather stations can forecast weather within the limited range of prediction levels such as sunny, rainy, cloudy, partly cloudy, stormy and snowy etc [6]. But other than purchasing a weather station, it can be developed a weather station using sensors and microcontrollers. Since microcontrollers are affordable to purchase, they can be a good development solution as an alternative to a commercial-grade consumer-level weather station. Also, since they are easy to replace, the easy maintenance of the weather station can be certified. With the advancement of technology, there are many

ways in which we can use to capture weather data. Nowadays, there are lots of weather-related sensors are available in the market [7]. On the other hand, those sensors are compatible with most of the popular Microcontroller Boards such as Arduino Controller Boards series, ESP 8266 and ESP32 Wi-Fi-based SoC boards and Raspberry Pi SBC series etc [8]. Therefore, using these, it can be built a weather station to measure and predict weather-related data.

1.2 Problem Statement

Commercially available weather stations are mostly unaffordable due to their high price tag. Most of those can measure temperature, humidity and barometric pressure. Based on the hierarchy of the product line-up, those are capable to measure wind speed, wind direction and rainfall. Furthermore, some can calculate heat index, dew point and wind chill. Some are capable to measure the maximum, minimum and average values of certain weather parameters. And, some higher-end models are capable to predict the upcoming weather up to a certain extent. When features are increased, the price of the weather station becomes much higher.

Considering the connectivities of the weather stations, most are used wireless communication methods. Among those 433MHz Radio Frequency (RF) based communication is popular. Some high-end models used Wi-Fi to transmit data to the indoor unit or used to transmit data to their IoT cloud storage.

Considering the power source that is used, those are battery-powered and most of them are powered by AA-size batteries. Therefore, those batteries cannot be used as rechargeable batteries. This also affects to increase the maintenance cost of the weather station. But most high-end devices are capable to power up the indoor unit with rechargeable batteries or outdoor units using a solar panel.

In the Sri Lankan context, weather predictions are limited to major cities only. Based on the Weather Map provided by the Department of Meteorology of Sri Lanka shows weather data only for 24 locations [9]. For some districts, there are no weather data on that map [9]. Therefore, based on that data, it is very difficult to get an idea about how today’s weather will behave.

In that kind of situation, most people have to rely on the closest city's weather forecast. Sometimes those forecasts may not be accurate for their area. This will leads to a requirement for a consumer-level weather prediction system Therefore, that problem will be addressed in this study. With this research, planned to develop a Microcontroller based Weather Prediction System with weather predictions and a remote data logging facility which can be observed in a high-end consumer-level weather station. Also, the accuracy of the predictions will be checked based on the given results from the system and actual data.

1.3 Aims and Objectives

The main aim of this research is to develop a microcontroller-based Weather Prediction System which can be used to predict the weather data and store that data on a remote server. This weather station will monitor weather-related data such as temperature, humidity level, atmospheric pressure, wind speed, wind direction and precipitation etc. microcontroller board will be used to control the data capturing and storing due to its low power consumption and affordability.

The objectives of this research are:

1. To design a microcontroller-based system to measure weather data.
2. To use the microcontroller to predict upcoming weather conditions.
3. Store the captured data in a remote database.
4. Provide a dashboard for displaying captured data.

1.4 Proposed Solution

From this research, planned to predict the weather using weather-related sensors and Microcontroller Boards. Historical data captured by those sensors will use to predict the weather. Also, historical weather data will be stored in a remote server to fine-tune the prediction model. Devices will be powered up using batteries with solar power charging as a sustainable energy solution. This kind of weather prediction system is more useful to rural areas in Sri Lanka due to the less availability of weather-related resources. Since we are receiving sunlight for the entire year, solar energy power-up will be a greater solution.

Considering the features of the weather prediction system, it will contain measuring of temperature, humidity, atmospheric pressure, precipitation, wind speed, wind direction, dew point, heat index, sunrise time, sunset time, solar noon time, moon phase and upcoming weather prediction. Also, this will have interactive user interfaces showing the current readings and a detailed view showing minimum, maximum and average values of certain weather parameters. And to save energy, this has brightness controlling and screen timeout options. Considering the outdoor unit, this will be powered by using solar panels and rechargeable batteries. The indoor unit is capable to send the captured weather parameters to the Adafruit IO IoT cloud service every 10 minutes.

1.5 Structure of the Thesis

The structure of this thesis is as follows. The first chapter will be discussed the background of the study. Also, it discussed the problem and how it will be addressed. Then the aim and the objectives of this study will be discussed. Then the second chapter will be discussed the related studies done similar to this research. The third chapter will be discussed the technologies that were adapted to this study. The fourth chapter contains the methodology of the study and it will be described what will be used to develop this system. The fifth chapter contains a brief description of how the system is designed and the main modules are functioning. The sixth chapter will be discussed how the system is implemented, how codes are used to process the logic and the related diagrams of the system architecture. The seventh chapter contains the evaluation of the study and it described whether the study is achieved the research objectives mentioned in the first chapter. Finally, the eighth chapter contains the conclusion of this study and the proposed future works.

Chapter 2 - Literature Review

Through this kind of weather station, we can measure weather-related data such as temperature, humidity, atmospheric pressure, precipitation, solar intensity, wind direction etc. to capture those data, multiple sensors can be used. Based on this article [10], to capture temperature and humidity, they used DHT11 and LM35 sensors. Also, this article [11] uses DHT11 to capture the same. When considering about accuracy level of DHT11, it can measure relative humidity up to 20% - 90% only. A country like Sri Lanka may have more than 90% of relative humidity [12]. Also, it has an accuracy of 5% [13]. Therefore, using DHT11 will give inaccurate values. In the DHT sensor series, there is a more accurate sensor called DHT21. In this research [14], they used DHT 21 sensor to capture temperature and humidity. Due to its high stability and reliable readings, this would be a great selection for temperature and humidity sensors.

When considering capturing atmospheric pressure, there are two common options. There can be used BME280 or BMP180 sensor. In this article, they [15] use BMP180 and they [16] used BMP280 which gives the same results when compared to atmospheric pressure. But BMP180 sensor does not have humidity measurement capability and low resolution for temperature when compared to the BME280 sensor [17] [18]. Based on the datasheet of the BME280 sensor, it can be used to capture temperature, humidity and atmospheric pressure together and it helps to save some Input-Output (I/O) pins in the microcontroller and save space other than using two sensors [18]. On the other hand, if accuracy plays a major role in research, using high-accuracy sensors would be a great advantage. To capture atmospheric pressure, MPL3115A2 would be more accurate than the BMP sensor range. And this article [19] used the same sensor for their low-cost weather station. Also, it is the default sensor that comes with SparkFun Weather Shield [20]. Based on the datasheet [21], this sensor has an accuracy of 30cm change in altitude in high precision mode. Since this was recommended for weather data capturing, choosing this sensor in research would be a great advantage on the precision side.

To capture the rainfall, multiple methods can be used. Using a raindrop sensor, we can identify whether it is ringing or not. This weather station [22] uses that technique to capture rainfall. Therefore, using this method, it is not possible to identify the rainfall amount. Therefore, for a weather station, this could not be an ideal solution to capture the rainfall. Also, rainfall can be measured using a sensor. The Tipping Bucket method is mostly used for that. This article [16] used the tipping bucket method to measure the rain for their weather station project. This sensor used a hall effect sensor to measure the movement of the tipping bucket. Figure 3.1 shows [23] the structure of the tipping bucket rain gauge. Number one shows the funnel that guides the raindrops to the tipping

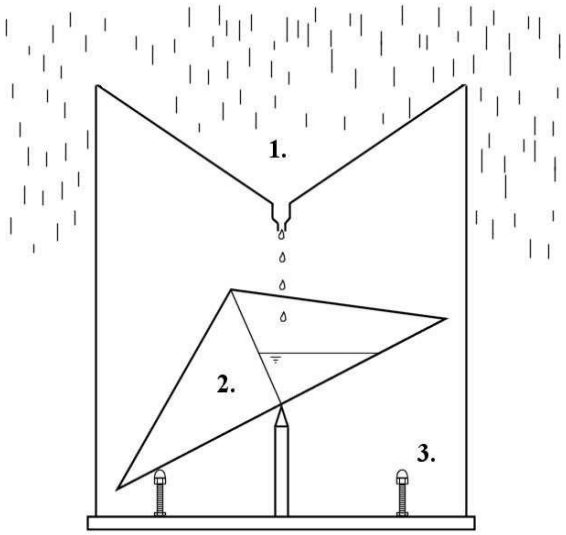


Figure 2.1: Tipping Bucket Rain Gauge
Source: adapted from [23]

bucket. Number two shows the tipping bucket and it will collect the rain drops coming through the funnel. After filling one side of the tipping bucket, the rainwater will drop due to gravity. There are adjustment screws shown by number three that can be seen in some tipping bucket rain gauges to calibrate the tipping bucket. Due to this continuous movement of the tipping bucket during the rainy situation, based on the movement counts, it can be measure the precipitation level.

To measure the wind speed there can be used an anemometer. To measure the speed of anemometer rotation, there can be used IR obstacle avoidance sensor like this article [24]. Or else, it can be used hall effect sensor and magnet to measure the same. Using a hall effect sensor is much easier. This article [25] used a hall effect sensor since it required a small space. Measuring the rotations count can be used to measure the wind

speed. Based on the pre-defined wind speed for one rotation per period can be multiplied by the current rotations per period will give the wind speed. Other than using a hall effect sensor, reed switches are also a possible cost-effective method to measure the rotations per second.

Measure the wind direction, there can be used very complex direction sensor based on MEMS sensors like the one developed in this article [26]. For that, they used polycrystalline silicon film to develop the sensor. Or else can be used simple sensor developed based on the reed switches etc. like one used here [16] for weather monitoring. Figure 3.2 describes the internal structure of the wind direction meter. It

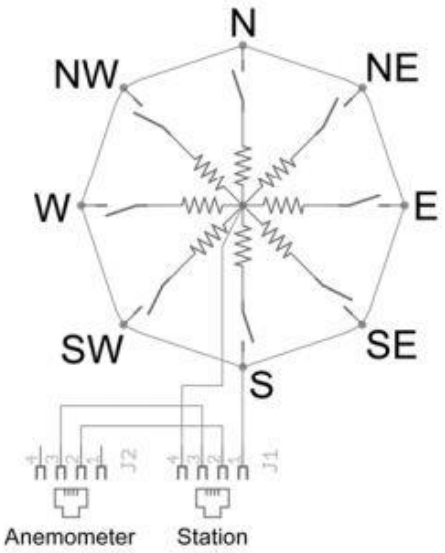


Figure 2.2: Vind Vane Meter
Source: adapted from [47]

has 8 reed switches installed with every 45 degrees of angle. Each switch is connected to a resistor that has different resistance values. When the magnet reaches a reed switch, it will be closed and the electricity passed through the switch and the resistor. Therefore, based on the resistance value given by the wind vane meter, it can be measured the direction of the wind. Since this magnet is capable to closed two proximity reed switches, the resolution of wind direction would be 22.5 degrees. Therefore, it is capable to measure 16 directions.

In a weather station, data logging is a major process. All predictions are based on those data. If the weather station has an only outdoor unit, then the data logging mechanism should be integrated into the outdoor unit itself. But if that unit has both indoor and outdoor units, it is better to have a data logging mechanism attached to the indoor unit.

Some outdoor units used GSM modules [27] to transfer data via SMS [27] [28] or through GPRS [29] to a remote server. The major disadvantage of the GSM module is its high power consumption compared to other modules. Some modules have a peak power of 10W (5V 2A) to maintain a GSM connection. Therefore, GSM-based communication is not suitable for most outdoor units if the unit uses battery power. But this is more suitable for the indoor unit since the user can provide the required continuous power to the module. Sometimes, they used long-range RF transmission to send data to another location and from there, further processing or data logging will happen. Since most of those communication devices consume less power than GSM modules, those can recommend for outdoor battery-powered units. Another method is using ZigBee-based protocols like this article [30]. Also, there is another method that can send data for long-range which is called LoRa-based protocols. If the data logging is done in a remote location, this LoRa-based protocol is more useful for research works like this [31]. If you require to send data to medium range nRF24L01+ module will helpful like this article [32]. That module has very low power consumption but can send data to a certain range which is helpful for most wireless communication projects. Also, one module that can listen to 6 channels simultaneously and up to 800 meters of range with the Power Amplifier (PA) and Low-Noise Amplifier (LNA) will become very useful in weather-related projects.

In a project like a weather station, visual representation is more important. Other than just showing some values, if we can display the values in some graph, gauge or scale it would be easy to understand. Therefore, a display unit in a weather station plays a major role on the user side. For this kind of project, most researchers [33] [34] use an HD44780-based character display. Those displays have come in different sizes such as 16x2, 20x4, 16x4 etc. use of these displays is very easy. But if we need to display something attractive and allow user interactions, the best solution is touch-enabled Thin Film Transister Liquid Crystal Displays (TFT LCDs). But drawing something on those displays is very complex. Therefore, there is a special display called Nextion Displays [35] which has an integrated microcontroller to control the display itself. Therefore, if someone needs a colour display with touch interactions with ease of use, it is very easy to use that kind of display for that kind of requirement. Because it has separate software to design the UI [36] and only we need to receive and transit required values to the

display. The display will handle the rest. Using this kind of display for a weather station [27] would be very attractive to the user regarding getting predictions from it.

If the project plans to with the support of a microcontroller, we have to select the most suitable microcontroller for it. Most weather stations' used Arduino-based microcontrollers [27] [10] [28] since their availability, price, supported sensors and availability of libraries for developing the programme. But if the project required additional communication facilities like Wi-Fi, dual-core processor, Digital to Analogue converter pins etc., selecting an ESP8266 or ESP32 microcontroller board would be a great choice. Projects developed with those microcontroller boards [11], would have additional advantages like high processing power, huge Static Random Access Memory (SRAM) and other memory spaces to do more complex works [37]. But if you need more than that like machine learning to analyse the collected data, Raspberry Pi-based SBC can be used for it. These projects are done by using Raspberry Pi SBCs, One uses normal data logging [38] and the other one used Machine Learning using Python language [38].

In a weather prediction system, a prediction algorithm has more value than any other thing. Every prediction's accuracy depends on the quality of the algorithm. There are several algorithms used to predict the weather. This article [39] uses an algorithm developed by John B. Young [40]. Based on this algorithm, it can get a few predictions about atmospheric pressure status, rain status etc. Sometimes, Machine Learning can be used to predict the weather. In this article [11], they use Logistic Regression to predict the weather status. Also, this research [41] used Support Vector Machines to train the prediction model. And this article [34] used Sliding Window Algorithm to predict the temperature. On the other hand, this article [42] used multiple non-adaptive and adaptive prediction algorithms to predict the Temperature, Humidity, Wind speed, Wind Direction, Rain, Pressure, and Light intensity. When considering rainfall, this article [43] predicts the rainfall by using various prediction algorithms and displays all results. And this article [44] used Deep Reinforcement Learning algorithms to predict the weather. This article [45] uses Zambretti Algorithm to forecast the upcoming weather status.

Chapter 3 - Technology Adapted

3.1 Controller Board

To capture the weather-related data from sensors, there are many methods are available. Most industrial-level sensor manufacturers have systems to capture the data from their sensors. Considering the prices of those modules can be expensive and require technical skills to integrate with the other systems or to build a sensor network. Therefore, microcontrollers can be considered an affordable method to read the data from the sensors. Since most of those sensors including cheaper sensors to industrial-level high-precision sensors are compatible with communication methods that are used by microcontrollers. Universal Asynchronous Receiver-Transmitter (UART), Serial Peripheral Interface (SPI) and Inter-Integrated Circuit (I2C) are most commonly used in microcontroller's data communication methods. Since most sensors are compatible with those data communication methods, selecting a microcontroller would be an affordable method to capture data from weather-related sensors.

Among microcontroller-based development boards, there are the most popular microcontroller-based development boards available in the market. Arduino controller boards are can be considered the most popular microcontroller-based development boards among those. The first Arduino controller board was introduced in the year 2005 [46]. During those 16 years, it became more popular among students, makers, hobbyists and engineers etc. to implement their creative ideas into a working prototype or sometimes into a commercial product.

The Arduino controller board is simple and it won’t make you confused and doubtful while prototyping [47]. Also, the Arduino development board is very cheap and easy to clone since the design is open-source [47]. Therefore, you can develop your Arduino-compatible board or change it as per the requirements of the prototype. When considering learning time, it is very easy and less time-consuming due to the thriving community [47]. Hence, it is easy to find solutions for the issues rises while prototyping. Also, Arduino has their own Integrated Development Environment for programming the development boards. The programming language was developed based on C and C++ [48] and there are thousands of libraries that can be used to simplify the programming while working with sensors and modules.

Arduino has many development boards manufactured based on different requirements with different form factors. In their product series, Arduino Uno and Arduino Nano are the most popular in the Entry Level category [49]. These boards are popular among those who starting to explore the Arduino world. Also, Arduino Leanardo and Arduino Pro Micro are famous because of their capability to emulate keyboard or mouse inputs by using the USB HID protocol [50]. This feature is more useful in some development for use macro key programming or executing repeated keyboard or mouse inputs etc.

In the Enhanced category, Arduino Mega 2560 is the most popular one [49]. That is one of the high-end products that they manufactured under their main development board series. Since this board has many I/O pins and larger memory, it is more suitable for larger-scale prototyping. But, other than that, there are many new development boards and expansion modules/ shields available. Currently their IoT product range and Pro series are becoming popular [51]. With the newly introduced Arduino Pro series, they released development boards from the Portenta Family, MKR Family and Nano Family and a lot of modules and expansion boards compatible with that. Those products are mostly centred on IoT, industrial-level applications and AI-based implementations [51]. This is a good sign that we can predict the Arduino platform will go towards the industrial-level solution-providing platform in near future.

Other than Arduino controller boards, there are popular microcontrollers manufactured by Espressif Systems. ESP 8266 and ESP 32 are the most common of them. Considering the specifications of the ESP 32, it has 240 MHz dual-core 32-bit microprocessors. And 512kB of SRAM, 448kB or Read-Only Memory (ROM), 12-bit Analogue to Digital Converter (ADC) pins up to 18 channels, 8-bit Digital to Analogue Converter (DAC) pins up to 2 channels, touch sensors up to 10 channels, 4 SPI channels, 2 Inter-IC Sound (I2S) channels, 2 I2C channels, 3 UART channels, hall effect sensor, 16 Pulse Width Modulation (PWM) channels, Bluetooth 4.0 connectivity and Wi-Fi 802.11 b/g/n standards etc [52]. Also, this can be programmed with the Arduino IDE and using MicroPython, this can be used for lots of applications such as low-power IoT sensor hub, data loggers, ESP32-CAM module for video streaming, speech and image recognition, home automation, industrial automation, audio applications, health care applications, wearable electronics and retail and catering application etc [52].

Based on the nature of this study, the indoor unit of the weather station will be powered by an ESP32 microcontroller-based development board. This was selected due to its Wi-Fi connectivity, larger memory to process more data and processing power. The outdoor unit will be powered bu the Arduino Mega 2560 microcontroller-based development board. It was selected due to the compatibility of sensors and most of the sensors are 5V logic-level supported. Also, it can handle 3.3V logic-level devices smoothly too. Also, the availability of the libraries helps to perform the coding process more easily and well organised.

3.2 Prediction Algorithm

Since this project is powered by only microcontrollers, the weather prediction algorithm should be a less resource consume one. By analysing the algorithms that are used to predict the weather mentioned in chapter 2, most of those required high processing power and lots of memory capacity. But considering the Zambretti algorithm, using the atmospheric pressure weather prediction would be possible. This algorithm was introduced in 1915 by precision instrument makers Negretti and Zambra. It was a circular shape device that was able to predict the weather for up to 12 hours. To predict the weather using this algorithm, first, the current atmospheric pressure has to be converted to the sea level pressure using a formula. Then the trend of the atmospheric pressure change has to be analysed. Based on the trend, there is a specific formula to calculate the Z value. Then the effect of the wind has to be added to the Z value. Later, based on the final Z value, the weather prediction can be determined.

3.3 Communication Mechanism

Since this weather prediction system has 2 units called indoor and outdoor, there should be a good communication method between those two units. In a domestic environment, there are walls and concrete structures surrounding. Therefore, a wireless communication method would be convenient to use. Among popular wireless communication methods like Bluetooth, 433MHz RF transmitter and receiver, Wi-Fi-based communication etc. nRF24L01 module was selected to manage wireless communication. This module used 2.4GHz frequency which is reserved for industrial, scientific and medical purposes internationally. Also, this can transmit data up to 2 Megabits per second (Mbps) speed. And the modified version nRF24L01+ PA LNA

module has 800 meters line of sight range. Therefore, short distances with considerable obstacles would not be an issue for wireless communication using these modules. On the other hand, this can act as a transmitter and a receiver, two-way communication is possible with this module. as per the nature of this system, the outdoor unit's module will work as a transmitter and the indoor unit's module will work as a receiver.

3.4 Energy Source

There are multiple ways to power up the indoor and outdoor units. The easiest method is providing power using the main electricity supply through a voltage converter. Since both units can be powered up by using 5V, any mobile charger can be used to power up the devices. The other method is to use batteries. Both units have an internal voltage regulator, we can use 9V batteries or can series the 1.5V batteries up to 4 units. Since the prevailing issues in the country, using batteries or the main electricity supply would

Figure 3.1: Solar Panels Arrangement

not be a great power source for this project. Also, using rechargeable batteries are save money which will be wasted on replacing the batteries. But since those are has to be recharged, it has to be used main electricity supply generally. On the other hand, Sri Lanka is a country that is closed to the equator. Therefore, we have sunlight throughout the year. Therefore, solar energy would be the best source to power up these devices. Since solar energy is completely free, there is only cost that we have to bear is the cost

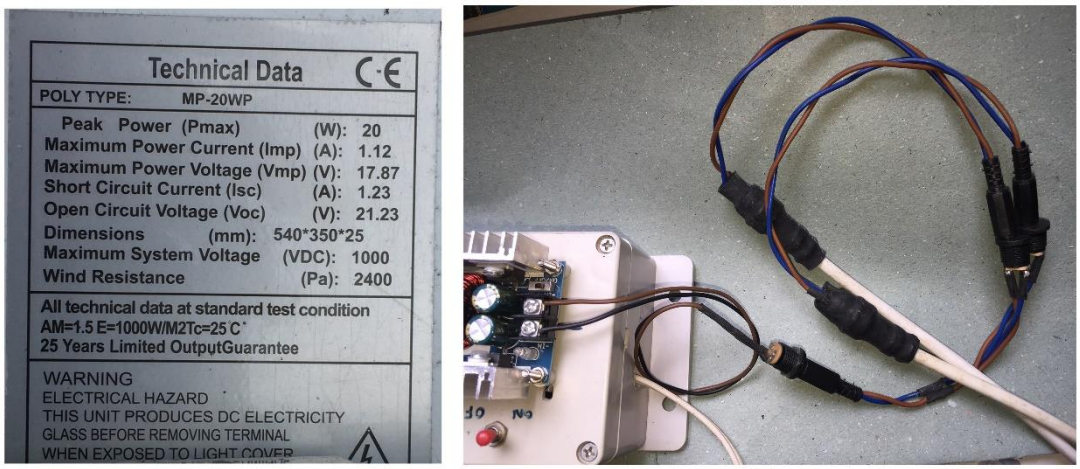


Figure 3.2: (a) Panel Specifications (b) Wire connection to Buck Converter

of solar panels. Since both units required 5V, there is no requirement for larger solar panels. A maximum 20W solar panel is enough for this kind of project. To get the maximum sunlight two solar panels were used in this study. Figure 3.1 shows how the panels are arranged.

Those placed facing opposite sides get the maximum level of sunlight to generate electricity. Since the direction of the sunrise and sunset are changing during the year, this kind of panel installation helps to arrange the solar panels manually to better align with the sun. Also, other than placing the panels horizontal to the ground, it was slightly angled using the built-in stand. Therefore, arranging like this helps to get the maximum efficiency. In the mornings, the panel facing towards the East direction will generate more electricity. In noon. Both panels are generating electricity equally due to the angled placements of the panels. In the afternoon, the panel facing towards the West direction will generate more electricity. That generated electricity was stored in batteries to power up the devices at night. At the end of the wires, both panels were connected parallelly (Figure 3.2 part b) and they will be connected to a buck converter (Figure 3.4 part a) to reduce the voltage. As per Figure 3.2 part a, a single panel generates 17.87 Volts (V) when there is a load connected to it. Since the battery charging modules are worked at 5V, the direct output from the panels is reduced to 5V using the buck converter.

18650 batteries (Figure 3.3 part a) are used to store the electricity and with the range of 4.2V to 3.3V, they can store up to 2650 milliampere hours (mAh) of capacity. This

battery can hold a maximum of 4.2V and not should be discharged below 3.3V for the safety of the battery. To charge the 2 batteries, there are multiple methods can suggest. One method is charging per battery using a TP4056 Lithium-Ion single battery charging module. The other method is charging a series of batteries using a Battery Management System (BMS). There are 2-cell and 3-cell BMS suitable for this project. But in the local market, there is no BMS available for 2-cell currently and even 3-cell BMS is available, none of those has all protections such as balance charging, over-charging protection, over-discharging protection, over-current protection and short circuit protection etc. Therefore, in some way, it could damage the batteries eventually.

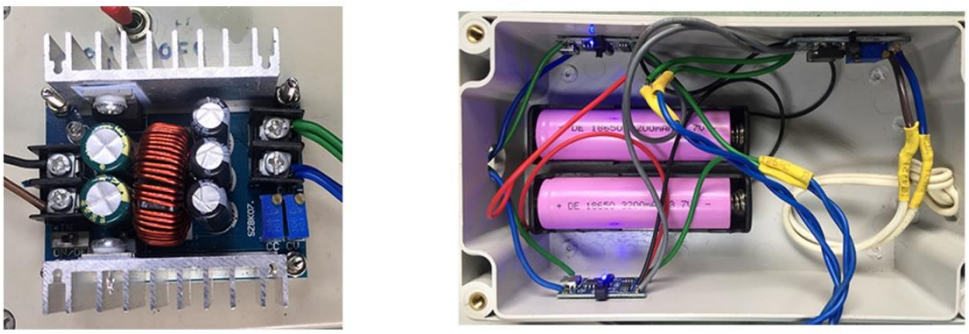
Also, the above-mentioned charging methods have 2 stage voltage conversions for powering up the Arduino controller board. If BMS is used, it has to use a Buck Converter to step down the voltage coming from the solar panels for charging and is required to use another Buck Converter to step down the voltage coming from the BMS for powering up the Arduino controller board.

If the TP4056 module is used, it has all the protections mentioned above and it has to use a Buck Converter to step down the voltage coming from the solar panels for charging and is required to use another Boost Converter to step up the voltage coming from the charging module for powering up the Arduino controller board.

(a) (b) (c)

Figure 3.3: (a) 18650 Batteries (b) TP4056 Battery Charging Module (c) Boost Converter

Therefore, by considering the above facts, TP4056 Lithium-Ion single battery charging modules (Figure 3.3 part b) are selected for this study. Using 2 TP4056 modules helps to properly balance charging and the long-lasting battery life other than parallelly connecting 2 batteries to one module. Since this module has over-charging protection, over-discharging protection, over-current protection and short circuit protection, using this module will help to maintain the battery life for a longer period. The output connects parallelly and since it carries a maximum of 4.2V, it has to be boosted up to



(a) (b)

Figure 3.4: (a) 20A Buck Converter (b) Complete Power Supply Module

5V again. Therefore, a Boost Converter (Figure 3.3 part c) was used for that. Then the boosted voltage is transferred to the outdoor unit for its functioning. To maintain the batteries and monitor the voltages in the power supply module, an external battery charging and the discharging unit would be ideal. Therefore, the power supply module (Figure 3.4 part b) was designed outside form the outdoor unit.

3.5 IoT Cloud

This weather prediction system will share the collected data with an IoT cloud service. This can be used to preview the collected data online anywhere in the world and also help to analyse the historical data to obtain some relationships between parameters. Initially, the Arduino IoT cloud was selected for this purpose. Their free account can store data on up to 5 variables and data will be remaining for 24 hours. Since it has an online code editor attached to this service, coding from the web is also capable in this cloud service. Unfortunately, when running the service on ESP 32 module, the entire programme will be delayed until the data is synced to the cloud. Since the weather dashboard attached to the ESP 32 module has a date and time display, on average, the programme will be stopped for up to 5 seconds at each time. Later, tried the Adaruit IO IoT cloud service and as per them, their free account is capable of having 10 variables and able to retain the data for up to 30 days. Also, they limited 30 data points per minute which leads to the fastest refresh rate up to 20 seconds when using all the 10 variables. Since weather stations are not required to update the cloud service at that rate, data will be sent every 10 minutes. Also, this service is well documented and code samples and tutorials were maintained of better quality than Arduino IoT Cloud. Also, dashboard features such as types of data interpretation instruments like gauges, text, multi-line text, buttons, data streams, separators etc. and multiple variables per line graph are better than the Arduino IoT Cloud. The most important point is while running the

Adafruit IO IoT cloud in ESP 32, there weren't any lag or programme stop situations visible and it was running smoothly with other operations. Figure 3.5 shows the view of the dashboard generated using the Adafruit IO IoT cloud. It shows the temperature, humidity and atmospheric pressure in line 1. In line 2, the heat index and the dew point are shown. Line 3 shows the wind speed and the wind direction. Line 5 and 6 show the rainfall and the weather prediction accordingly. Lastly, it shows the line graph showing temperature, humidity and the heat index for the last 24 hours. The second graph shows the atmospheric pressure for the last 24 hours.

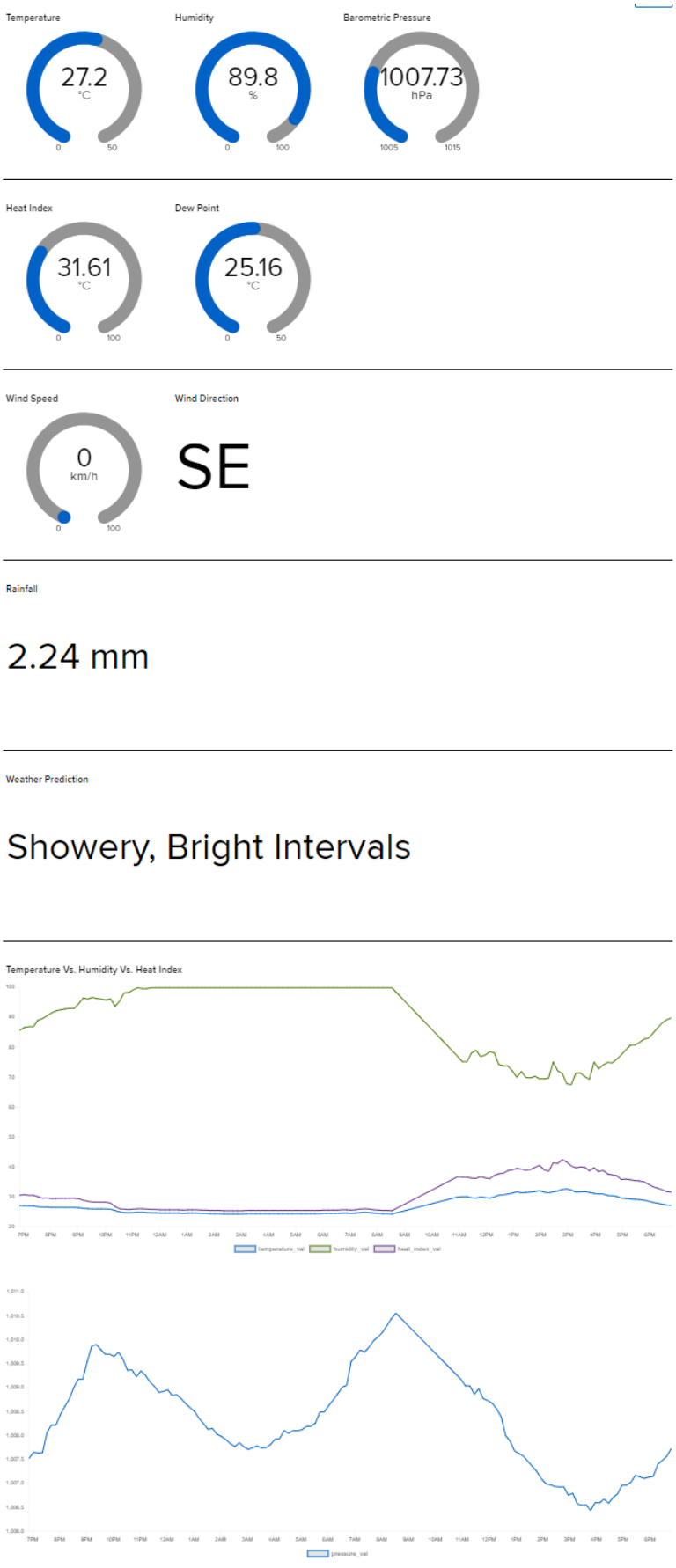


Figure 3.5: Adafruit IO Dashboard View

Chapter 4 - Proposed Approach

4.1 Overall System

As mentioned in the previous chapter, this weather prediction system will be controlled by microcontroller-based development boards. Therefore, the outdoor unit will be controlled by Arduino Mega 2560 development board. The indoor unit will be controlled by ESP 32 development board. This outdoor unit has sensors that are capable to capture weather-related data. For that, it will be used temperature and humidity sensor, barometric pressure sensor, tipping bucket rain gauge, cup type anemometer and wind vane meter. Also, there is DS3231 Real Time Clock (RTC) module will be used to reset the rainfall daily. Then the data will be transmitted to the indoor unit to process further. The indoor unit will analyse those data and maximum, minimum and average values will be calculated for some weather parameters. Also, using the atmospheric pressure and with the support of the Zambretti algorithm, upcoming weather will be predicted. Then using a Nextion display, those data will be presented as a dashboard view. Simultaneously, those data will be sent to an IoT cloud service every 10 minutes.

4.2 Components used to Develop Weather Prediction System

4.2.1 Arduino Mega 2560

Arduino is a platform that is popular among prototype developers, university students, electronic hobbyists and automation developers etc. Due to its popularity, most electronic sensors and components are manufactured with the compatibility of the Arduino development boards. Arduino Mega 2560 is their flagship device from the general product line. Figure 4.1 shows the top view of that development board. This board has 54 digital I/O pins, 16 analogue input pins, 8 kB of SRAM, 256 kB of Flash memory, 16 MHz of clock speed and 4 kB of Electronically Erasable Programmable Read-Only Memory (EEPROM) [53]. With the support of libraries developed for the sensors, this will capture the data from the sensors. Also, this board can handle both 5V and 3.3V logic levels, this can be used with any type of sensor. For this weather prediction system, I2C communication lines were used to communicate with the DS3231 RTC module. Since I2C can share the same 2 pins with many other I2C

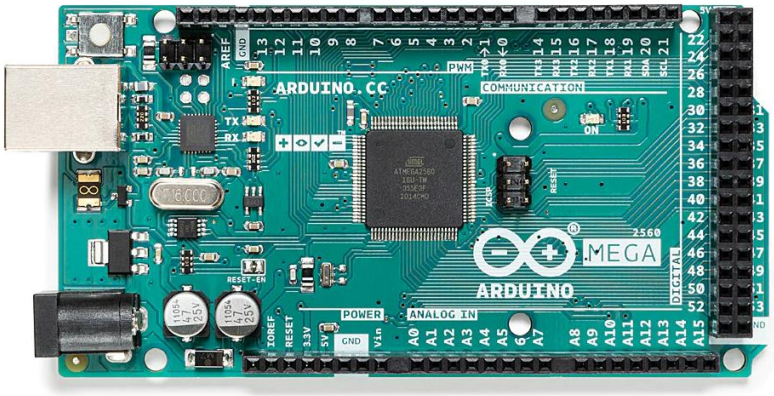


Figure 4.1: Arduino Mega 2560 Development Board

devices. The only this is the I2C address of each device has to be changed. In this development, the atmospheric pressure sensor and the RTC module commonly shared the I2C bus. Also, the SPI bus will be used for wireless communication. Since the nRF24L01 module used the SPI for communication with the microcontroller, SPI buses were used. For other sensors, it will be used digital pins to read the data.

4.2.2 Temperature and Humidity Sensor

To measure the temperature and the humidity, the AM2301 (Figure 4.2) sensor is used in this study. Other than DHT 11 sensor, this has more range and accuracy. DHT 11 sensor has a range of temperature from 0 °C to 50 °C and a range of humidity from 20% to 90%. Since Sri Lanka has quite a humid environment, DHT 11 sensor is not suitable for capturing the humidity. Considering the AM2301 sensor has a range of

Figure 4.2: AM2301 Temperature and Humidity Sensor

temperature from -40 °C to 80 °C. it can measure the humidity from 0% to 99.9% which is suitable for Sri Lanka's environment. Also, considering the resolution of the sensor, AM2301 has more resolution than the DHT 11 sensor. Since this sensor has a mounting hole also helpful for mounting it on a surface easily. This sensor only required one wire to communicate with the microcontroller. Therefore, this is a good selection for less wire usage since microcontroller development boards have a limited number of General Purpose Input/Output (GPIO) pins.

4.2.3 Atmospheric Pressure Sensor

To measure the atmospheric pressure, sensors manufactured by Bosch, Infineon and Freescale are popular among the community. Under Bosch sensors, BME280 and BMP180 are widely popular due to their affordability. But considering the resolution and the accuracy of those sensors, it is difficult to use in a high-precision weather prediction system. For a high-precision sensor, DPS310 which was developed by Infineon Technologies can be used and it has a maximum resolution of 0.002 hectopascals (hPa) and altitude can be sensed up to 2 cm precision. This sensor can be used for high accuracy requirements like weather stations, flight control boards in drones, indoor and outdoor navigation etc [54]. Also, there is another sensor MPL3115A2 which was developed by Freescale Semiconductor. This sensor has a maximum resolution of 0.015 hPa and altitude can be sensed up to 30 cm. This sensor also can be used for weather stations, navigation applications and smartphones or wearables etc [55]. Due to the availability of the sensors, MPL3115A2 (Figure 4.3) was

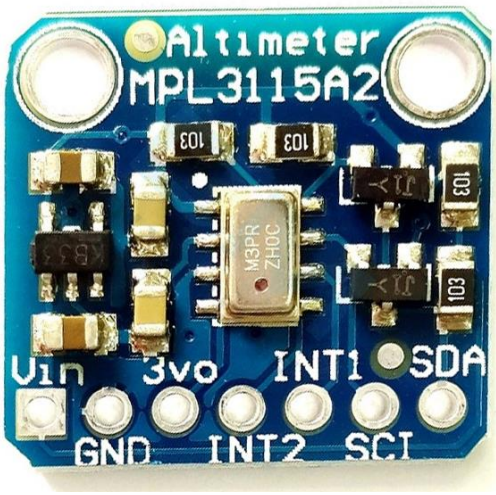


Figure 4.3: MPL3115A2 Atmospheric Pressure Sensor

used in this study. Since this is using I2C communication, it is easy to pair with the microcontroller. Measuring the atmospheric pressure is also easy due to the availability of the Arduino library.

4.2.4 Tipping Bucket Rain Gauge

To measure rainfall, there are multiple methods. There are dry sensors and wet sensors. In a dry sensor, they used the optical method to capture the rainfall. Those sensors have Infrared (IR) beams emitting from the transmitter to the receiver. If there is a raindrop, that will change the IR beam intensity due to escaping of some of the IR beams. Then the internal hardware and the software can identify the size of the raindrop. Considering the accuracy, this can detect a half millimetre raindrop even. Based on the model this accuracy become increased and these rainfall sensors have more resolution than tipping bucket rain gauges (Figure 4.4). Wet sensors are taking the raindrops inside the sensing mechanism. A tipping bucket rain gauge is a good example of that. Figure 2.1 shows the internal mechanism of this rain gauge. Once raindrops are falling into a bucket and after it is filled with rainwater, it will fall due to gravity. Based on that mechanism. The

Figure 4.4: Tipping Bucket Rain Gauge

movement of the tipping bucket can be captured using a magnet and a reed switch. The reed switch is installed in a fixed position of the sensing mechanism and it is located above the pivot point of the tipping bucket. The magnet is installed in the tipping bucket. Based on the sensor manufacturer, defined how much rainfall for a one-tip movement. Based on the tip movement count, the rainfall can be measured. Based on

this study, a tipping bucket rain gauge was used. Reed switches act like normal switches, but it is triggered when the presence of a magnet. Therefore, it has a ghosting effect on the output signal to the microcontroller which is known as bouncing. Thus, it creates inaccurate readings, it has to be eliminated by using a suitable method. One

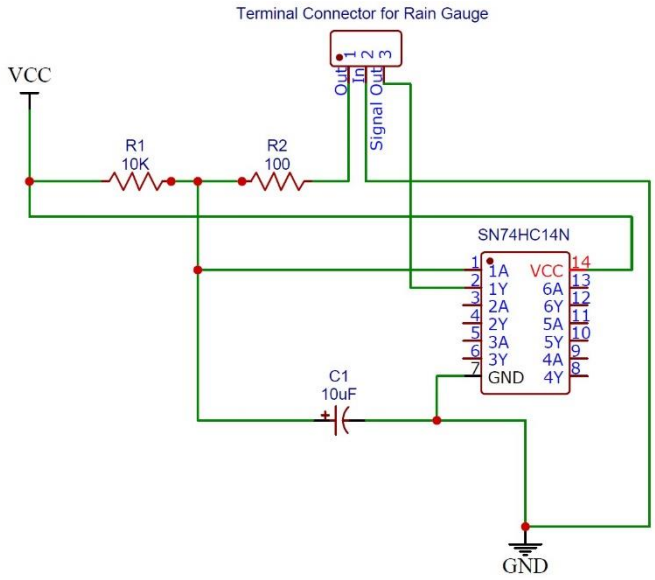


Figure 4.5: Schmitt Trigger Circuit Diagram for Rain Gauge

method is using a debounce time allocated in the microcontroller. But due to the nature of the movement in the rain gauge, this method is not suitable for a more accurate reading. Therefore, using the Schmitt Trigger would be a better solution for this bouncing effect. Since this is an external method to the main system, it reduced the additional resource utilisation of the microcontroller. Therefore, Schmitt Trigger is the best way to debounce the readings from the rain gauge. To create this circuit, SN74HC14N Hex Inverters with Schmitt Trigger Inputs IC [56] were used as the main component. Based on the circuit diagram shown in Figure 4.5, the circuit was developed on a breadboard, and the readings became more accurate. To capture the movement of the bucket, the Signal Out pin is connected to the interrupt pin in Arduino Mega 2560 development board. Therefore, an interrupt service routine is used to calculate the rainfall and it will reduce the resource utilisation since there is no requirement for continuous scanning of the signal out pin for a switch closed signal. As per the datasheet of the sensor, one movement of the tipping bucket contains 0.2794 millimetres of rainfall [57]. Based on the tipping count and per tip rainfall, the total rainfall is calculated.

4.2.5 Cup Type Anemometer

To measure the wind speed, cup type anemometer (Figure 4.5) was used. This has 3 cups and inside the moving part, there is a magnet installed. And the body has a reed switch which is activated by when the magnet is moving close to the switch. Based on the movement of cups, rotations per second can be measured.

Figure 4.6: Cup Type Anemometer

If the switch close once per second, the speed of wind is 2.4 kilometres per hour (km/h) [57]. even though the mechanism is as same as a rain gauge, using a Schmitt Trigger will give inaccurate readings when there are high revolutions per second. Therefore,

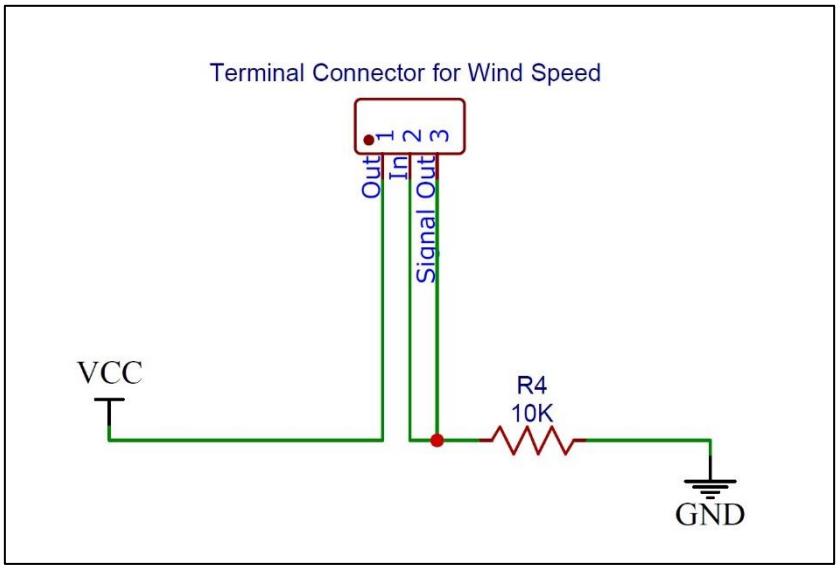


Figure 4.7: Circuit Diagram for Wind Speed

debounce time value is used inside the microcontroller. To keep the switch properly grounded, the circuit diagram shown in Figure 4.7 was designed and tested successfully.

4.2.6 Wind Vane Meter

To measure the wind direction, a wind vane meter (Figure 4.8) was used. This has a moving directional head with a magnet inside. In the sensor body, there are 8 reed switches arranged at a 45° angle. Figure 2.2 shows the reed switches arrangement inside the sensor. Also, these switches are connected to a resistor that has different values. Since the resistor values for each direction are pre-defined measuring the resistance, the wind direction can be identified. Since the magnet can be closed up to 2 switches when it is located in between two switches, using this meter, it can be measured up to 16 directions with 22.5° of resolution. Since the Arduino controller board cannot measure

Figure 4.8: Wind Vane Meter

the resistance directly, Voltage Divider has to be used for that purpose. Since it can read analogue input of 10-bit resolution, the voltage divider is the optimal solution to measure the resistance. As per Figure 4.9, the voltage divider circuit is designed. As the fixed value resistor 10 kiloOhm (kΩ) resistor was used and based on that value and the voltage received to the analogue pin of the Arduino board, the resistance of the wind vane meter can be calculated. Then, based on the datasheet of the sensor, using the resistance value, the wind direction can be calculated.

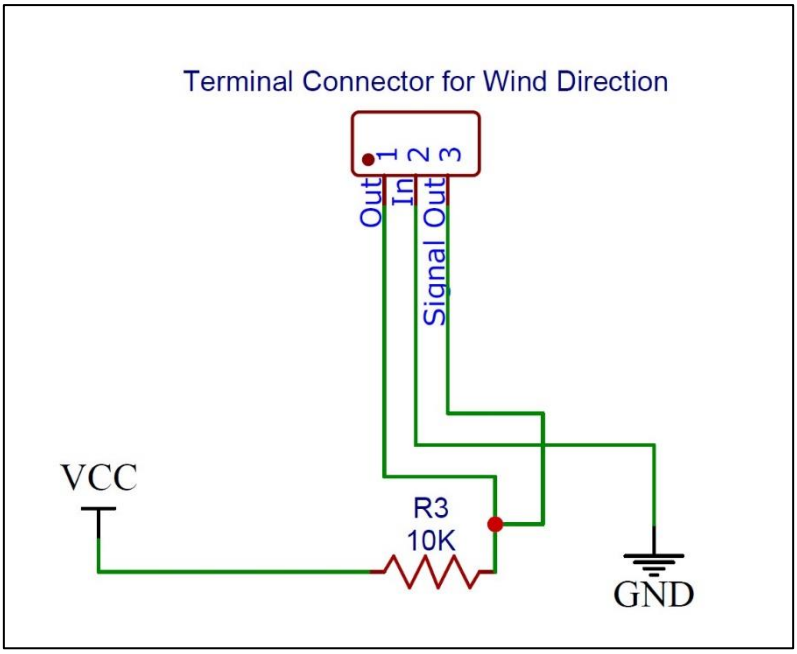


Figure 4.9: Circuit Diagram for Wind Direction

Since the rain gauge, anemometer and wind vane meter have an RJ11 jack to interface with the microcontroller, Figures 4.5, 4.7 and 4.9 combined and one circuit was developed in a perf board (Figures 4.10 and 4.11).

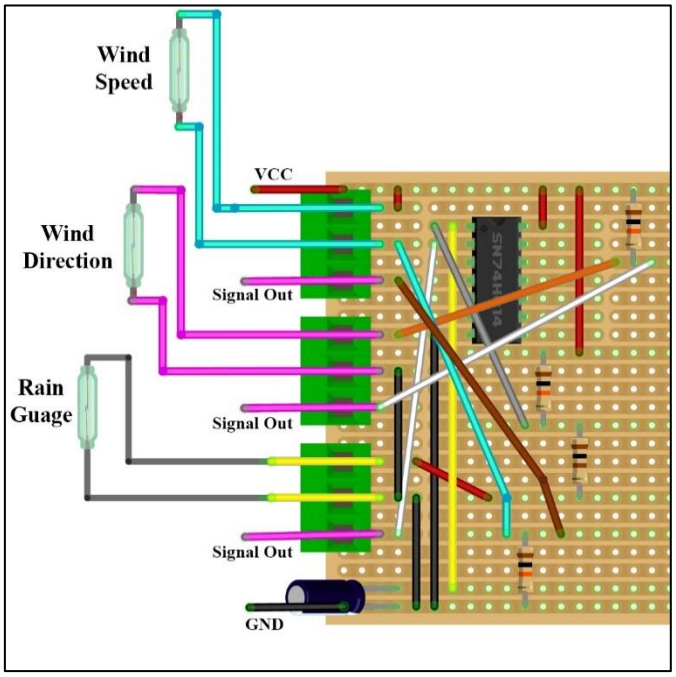


Figure 4.10: Final design for capturing rain, wind direction and speed

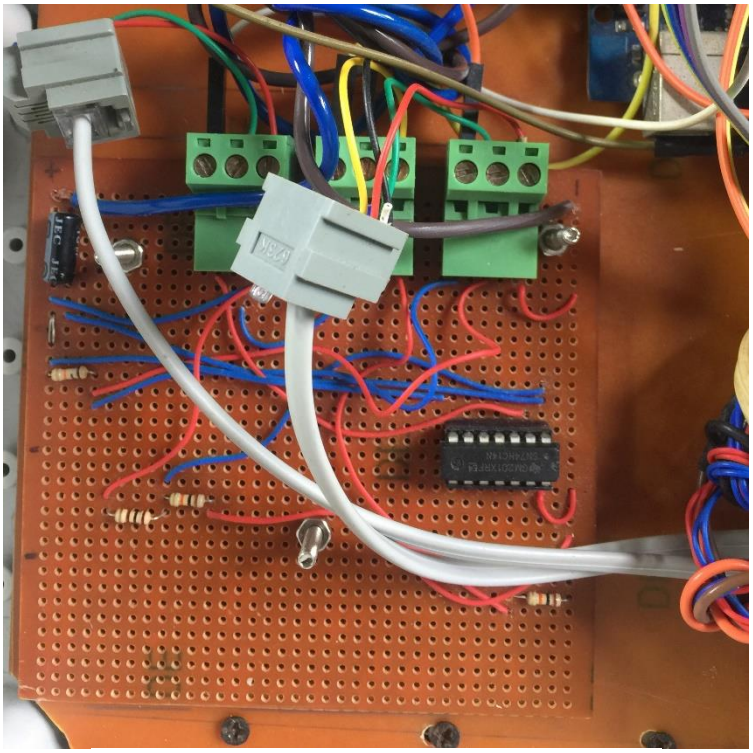


Figure 4.11: Actual implementation of Figure 4.10

4.2.7 DS3231 Ral-Time Clock Module

To keep track of time, RTC modules can be used. Since those have an internal battery, they can keep the time for a longer period without refreshing the memory. For this study, the DS3132 RTC module (Figure 4.12) developed by Maxim Integrated was used. Since this has more accuracy than peer modules, this was selected to continue with the study. Considering the DS1370 TRC module which has issues in keeping the

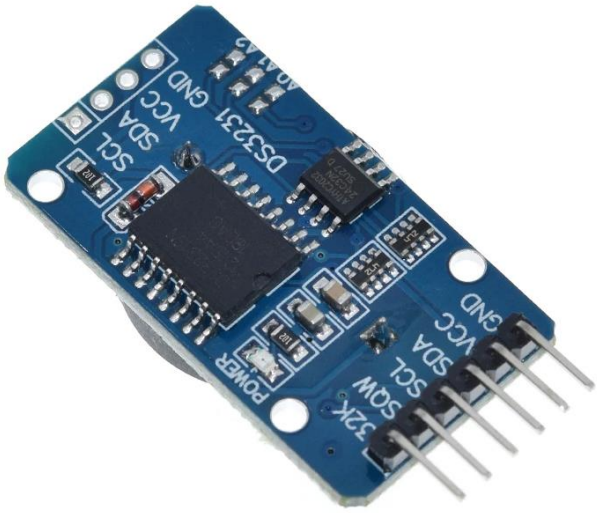


Figure 4.12: DS3231 RTC Module

time when there are temperature changes. Since DS3231 has Temperature

Compensated Crystal Oscillator (TCXO), that temperature issue can be mitigated. Therefore, the accuracy can be maintained for a deviation of a few minutes per year in the worst-case scenario. Also, this module has a CR2032 battery mount to install the battery. Due to very low power consumption (3 microampere) for timekeeping, this battery is enough for an approximately 8-year period [58]. The main purpose of this RTC module for this study is to manage the rainfall level. Since rainfall is accumulating, it has to be reset daily. Therefore, the date and time have to be maintained. Hence, based on this module's data, rainfall and the tip count will be reset when the date is changed.

4.2.8 Nextion Display

In this project, captured and processed data will be displayed. For that, the Nextion display (Figure 4.13) will be used. To display the results, there are many types of displays available. It can be displayed using a character display. But it would not be interactive to the user. Also, it can be used with TFT LCD which has a touch feature.

Figure 4.13: 5-inch Nextion Display

In the case of using a TFT LCD, all the interfaces have to be coded using the compatible programming language. Therefore, using a TFT LCD will be consuming lots of pins in the microcontroller, lots of work to design the User Interfaces (UI) and it will be a frustrating job. Since Nextion is also using a TFT LCD, it has a separate control board that required only 2 pins from the microcontroller other than power pins. The specific model used in this research is NX8048P050-011C-Y. This model has an 800*480 resolution, capacitive touch screen, 300 nits brightness display etc. also, this display has a 200 MHz processor to handle the incoming and outgoing data, 512 kB RAM, 128

MB flash memory, 8 GPIO pins, RTC module and audio interface. Since this has separate software to design and upload UI, it is very easy to create attractive UI. Therefore, the microcontroller only has to send the relevant data with the component ID, page ID and the parameter to the display. The internal microprocessor will process all those data and relevant fields will be updated. There are 16 pages designed for this project as user interfaces.

4.2.9 ESP 32 Development Board

To control the indoor unit, an ESP 32-S development board (Figure 4.14) will be used. Since this module has built-in Wi-Fi, it would be very easy to communicate with the IoT cloud. Also, the high-speed dual-core processor, larger RAM and flash memory helps to upload and run heavy programmes than an Arduino controller board. And, it

Figure 4.14: ESP 32-S Development Board

has a built-in RTC, it can maintain the time using a suitable library till it is powered up. To connect this to the other component, ESP 32 expansion board (Figure 4.15) developed by myself is used. The same expansion board was used in the outdoor unit to connect the power lines and I2C buses of multiple sensors. Using this kind of expansion board helps to manage the wiring and ensure the clean sensor and other component installation.

.

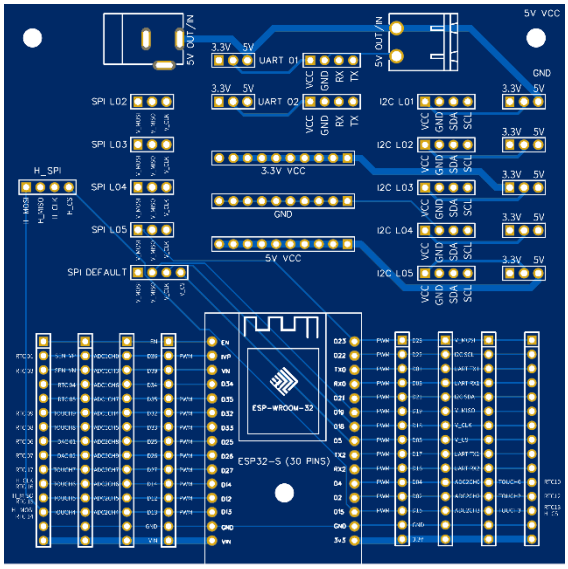


Figure 4.15: ESP 32 Expansion Board – PCB Design

4.2.10 nRF24L01 PA LNA Module

for wireless communication between indoor and outdoor units, the nRF24L01 module with PA and LNA (Figure 4.16 part a) is used. Since this has an amplifier unit for both receiving and transmitting, the range increased up to 800 meters. This module is very

Figure 4.16: (a) nRF24L01 with PA & LNA (b) Power Supply Module for nRF24L01

sensitive to voltage deviations. Therefore, it can be caused to maintain the data transmission range and the accuracy of packet transferring. To mitigate that, there is a separate module for supplying the power. This has capacitors to stabilise the voltage and we can supply 5V directly to this. Since the nRF24L01 is a 3.3V support device, the internal voltage regulator from the power supply module will reduce the 5V to 3.3V. Also, it helps to organise the pins in that module since the nRF24L01 does not have a pinout naming diagram on it.

Chapter 5 - System Design

5.1 System Architecture

The weather prediction system developed for this study has 2 main components. The outdoor unit handles all the data capturing processes using the related sensors mentioned in Chapter 4. And the indoor units handle all the processes regarding processing the received data, displayed in the dashboard and the upload to an IoT cloud service. Therefore, Figure 5.1 shows the overall system architecture of this weather prediction system. In that architecture, Figure 5.2 shows the weather data capturing unit of the outdoor unit. This unit is developed for a commercial-grade weather station and the hardware part can be used for other purposes too. There was a separate 433 MHz RF transmitter circuit inside this unit for transferring the data t their wireless indoor unit. But, that is proprietory and it cannot be decoded through a 433 MHz receiver, that unit is removed from the unit and DHT 21 (AM2301) and MPL3115A2 modules were installed. Then all the wiring is connected to the Arduino Mega 2560 development board. Then using the DS3231 RTC module, date changes will be monitored, if there is a date change, rainfall and the tip count will be reset to zero.

After all the data is gathered, using a float array, data will be transmitted to the indoor unit using the nRF24L01 transceiver module. Then the indoor unit is received those data using the nRF24L01 transceiver module, it will be processed and sent to the display for viewing purposes. Also, using a Wi-Fi router, ESP 32 is connected to the internet and sends the captured data to the Adafruit IO IoT cloud service.

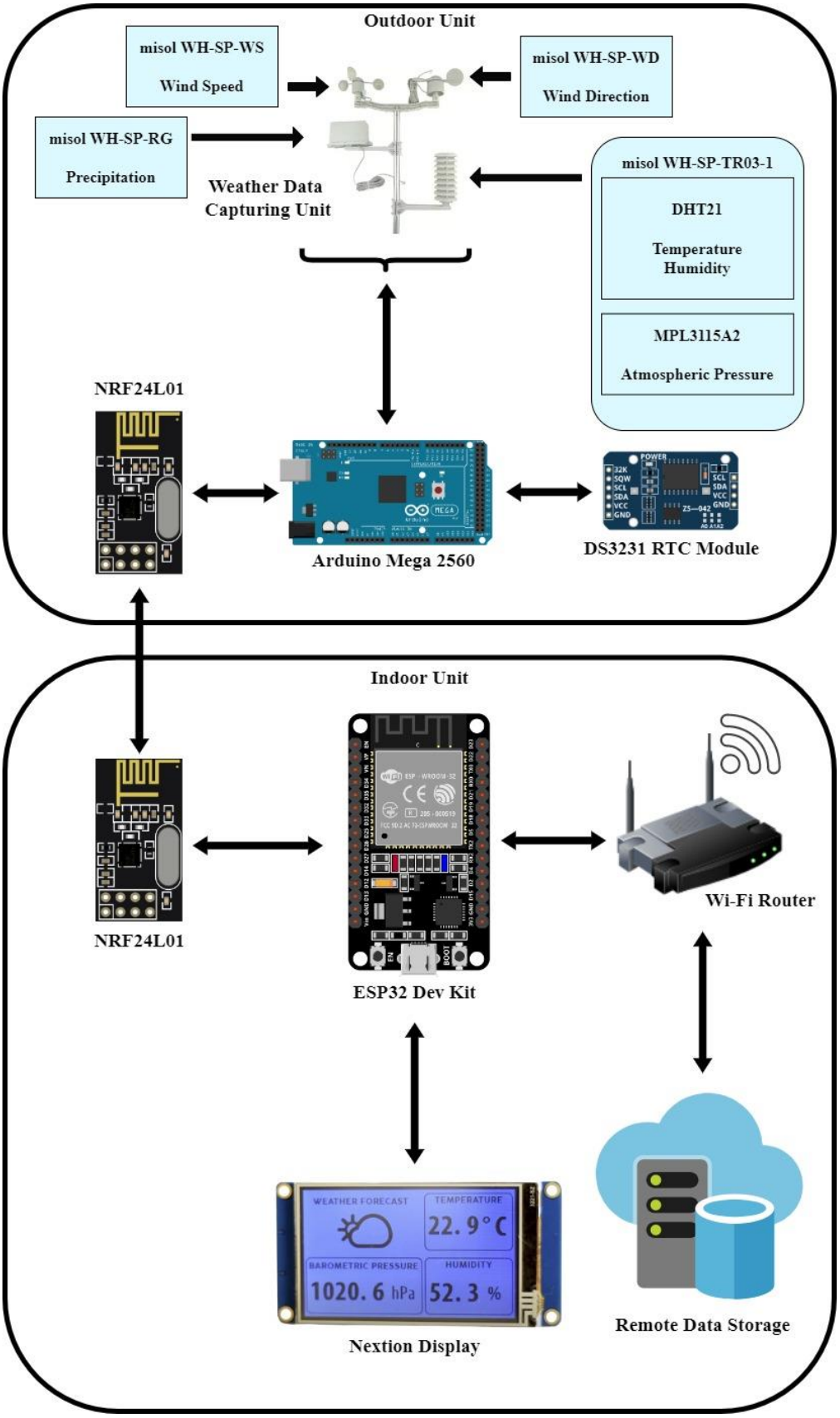


Figure 5.1: System Architecture of the Weather Prediction System

Figure 5.2: Weather Data Capturing Unit

5.2 System Working Mechanism

Figure 5.3 shows the flow chart of the outdoor unit. Here, first, all sensors are initiated. Then tip count, revolution per second, resistance from the wind vane meter, temperature, humidity and atmospheric pressure values will be captured. Then for precipitation, it will check whether is there any date changes happen. If yes, rainfall and the tip count will be reset. Otherwise, it will continue counting the tip count and using the per tip rainfall (0.2794 mm) and the tip count, total rainfall will be calculated. Then using the revolution per second and the wind speed for 1 revolution per second (2.4km/h) wind speed will be calculated. Then using the Adafruit library for DHT series sensors, temperature and humidity values will be calculated. Also, with the support of the same library, the heat index will be calculated. Then using the library for the MPL3115A2 sensor, atmospheric pressure will be calculated. finally using a for loop, minimum and maximum values for the voltage divider, wind direction in text and the angle will be calculated. Since the final array is in float data type, instead of text, the related index of the text array will be transferred to the final array. Finally, all those values will be collected together and transferred to a float array. Then it will be transmitted using the nRF24L01 module.

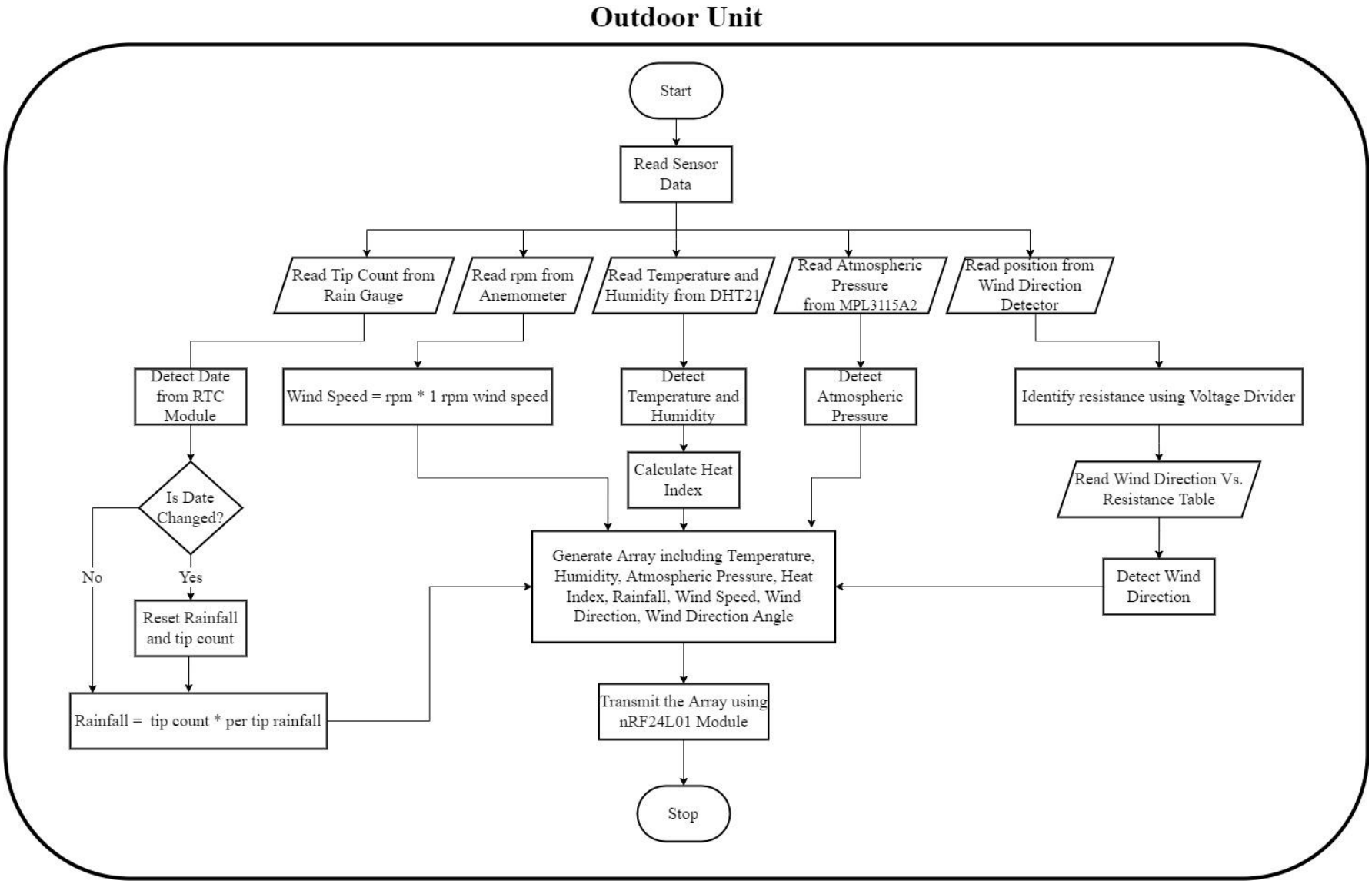


Figure 5.3: Flow Chart for Outdoor Unit

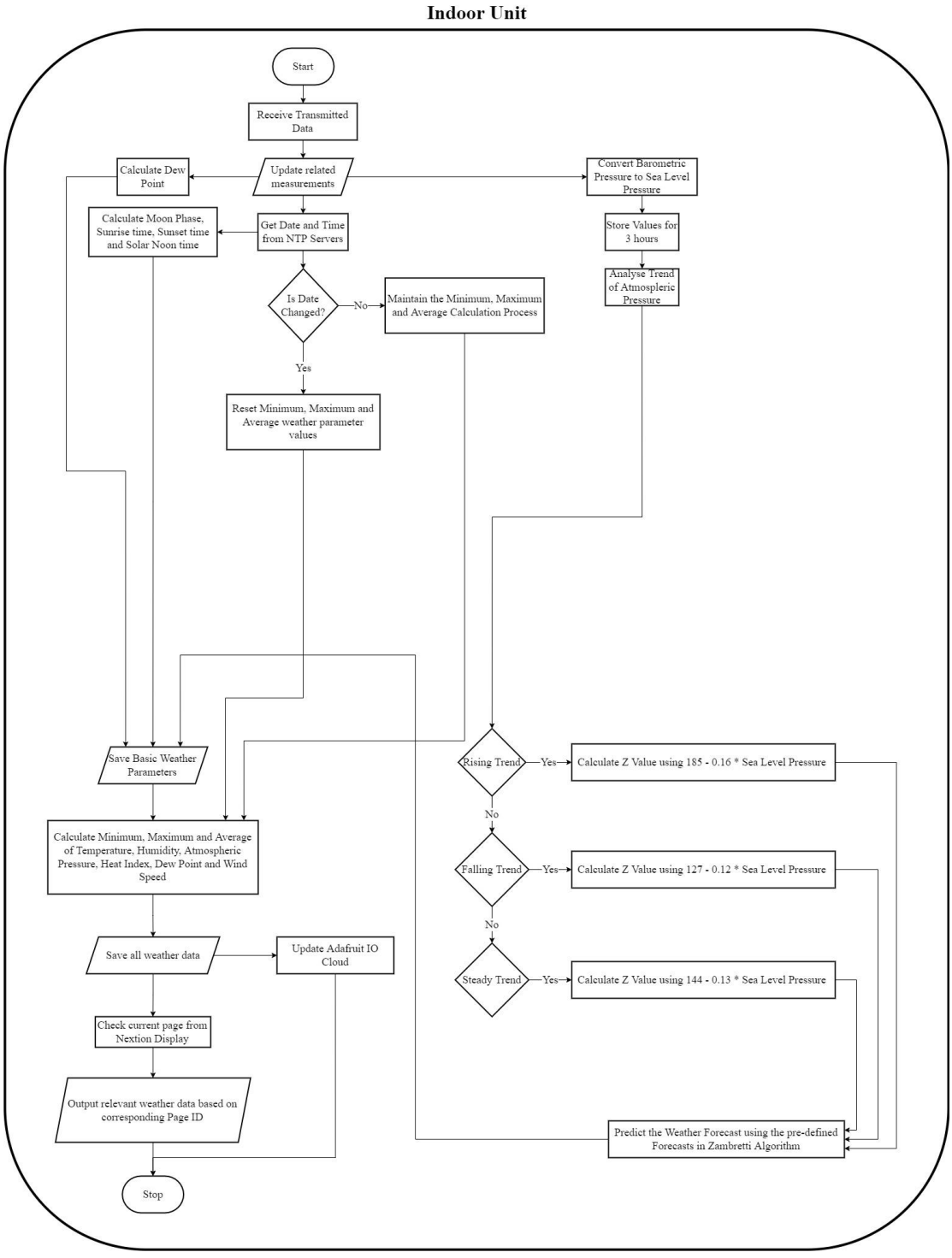


Figure 5.4: Flow Chart for Indoor Unit

Figure 5.4 shows the flow chart of the indoor unit. First, it will receive the data transmitted from the outdoor unit. Then it will allocate the received values to relevant variables. Using the below formula, the dew point is calculated.

$$\text{Dew Point} = \text{Temperature} - \frac{(100 - \text{Humidity})}{5}$$

Then, the date and the time will be received from the Network Time Protocol server (NTP). Then it will be stored and based on the date and the pre-defined location data of the installed location of the weather station, sunrise, sunset and solar noon time will be calculated using a library. Also, using the date, the moon phase will be calculated. Then using the current readings, maximum, minimum and average values of temperature, humidity, atmospheric pressure, wind speed, dew point and heat index will be calculated. Also, the date value that was captured using the NTP server will be used to reset the maximum, minimum and average values calculated.

Then atmospheric pressure will be used to predict the upcoming weather. For that, initially, the current reading of atmospheric pressure will be converted to the sea level pressure (SLP). For that, the below formula is used.

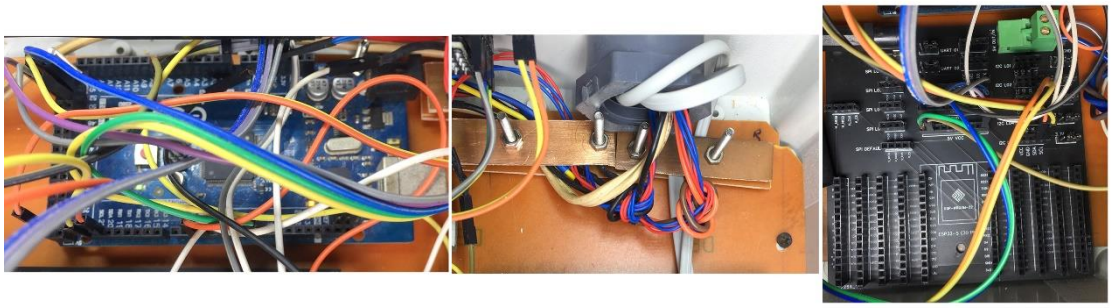
$$\text{SLP} = \text{Pressure}\left(1 - \frac{0.0065 \times \text{Altitude}}{\text{Temperature} + (0.0065 \times \text{Altitude}) + 273.15}\right)^{-5.257}$$

Then every 10 minutes, the SLP will be stored in an array. This array contains 18 values since the pressure trend has to be analysed for 3 hours. After the trend is identified, the current SLP has to be applied to a special formula based on the trend. Then the wind effect has to be integrated with the Z value calculated. Then it will be round to get an integer value for use as an array index. Then using an array, correct weather prediction will be extracted. After collecting all weather data, it will be sent to the Adafruit IO cloud every 10 minutes. Then, for the offline dashboard, relevant data based on the page ID will be sent to the nextion display.

Chapter 6 - Implementation

6.1 Outdoor Unit

Since the outdoor unit has to be installed outside the house, it has to be properly water-sealed. Therefore, a water-sealed project box was used for that. Then using the screw mounting holes in it, the copper board was mounted to the bottom since the water-sealed box cannot be drilled to mount the components. Then all the components are mounted to the copper board using M3 nuts and bolts. Also, 3mm plastic spacers will be used to increase the mounting height due to solderings underneath the components.



(a) **(b)** **(c)**

Figure 6.1: (a) Arduino Mega 2560 Development Board (b) Wire Tightening Mechanism (c) Expansion Board used to manage wiring

Figure 6.1 part a shows how the Arduino Mega 2560 development board is mounted to the copper board. Then part b shows how the wires are tightened to prevent any damage to the components from tensioning wires. Otherwise, if there is any tension applied to the wires, it may tend to get loose from the pin headers that connect the wire to the microcontroller. Then part c shows the expansion board that was developed for the ESP 32-S module is also used to manage the I2C and power buses.

Figure 6.2 part a shows the DS3231 module used in this project. And part b shows the nRF24L01 module plugged into the power supply module designed for it. Finally, Figure 6.3 shows how the components are arranged in the water-sealed box. Since the nRF24L01 module is attached using the cable tie, the holes drilled through the box were sealed using hot glue. The wires are taken inside using the 26mm hole. To cover it, Valve Socket and Faucet Socket are used with the rubber ring. For Valve Socket, a separate pipe was attached to prevent the splashing water from the rain. Then the water-sealed box was installed with the orientation of the wire intake facing downwards

(Figure 6.4). This installation strategy helps to prevent rainwater from coming inside of the box.

(a) **(b)**

Figure 6.2: (a) DS3231 RTC Module (b) nRF24L01 Module with Power Supply Module

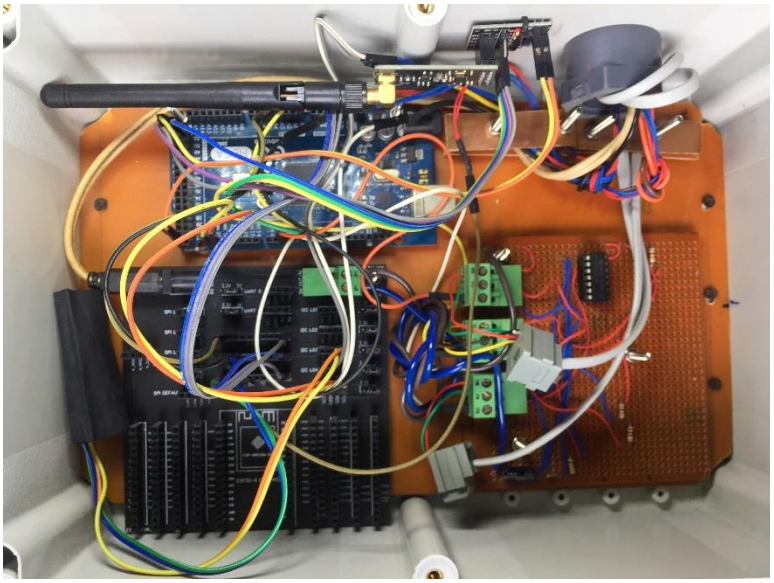


Figure 6.3: Outdoor Unit Components Arrangement

Figure 6.4: Outdoor Unit Installation

6.2 Indoor Unit

The indoor unit has only 3 components. The main component is the ESP 32 module and the expansion board. This unit handles all the data capturing, analysing, displaying and syncing to the IoT cloud. Its powerful processor and larger memory help to handle complex calculations. Then, there is the nRF24L01 module with the power supply

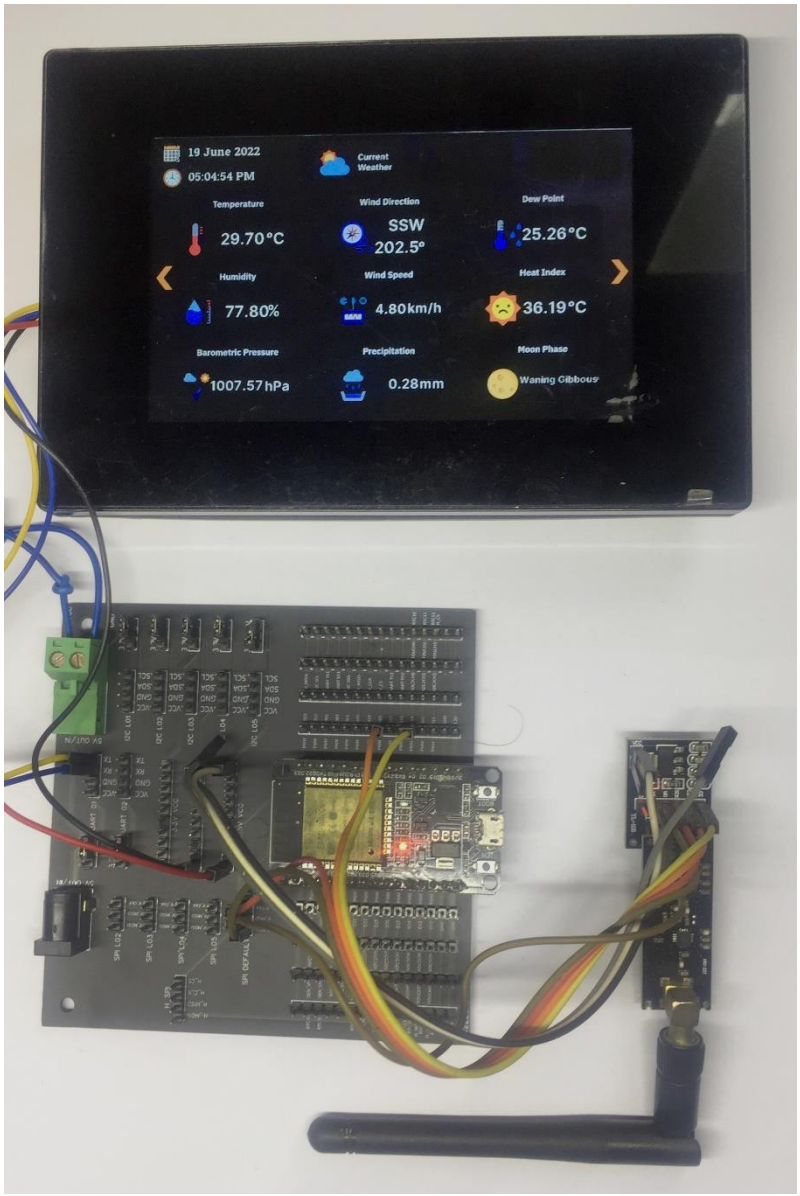


Figure 6.5: Indoor Unit Components Arrangement

module for receiving the data from the indoor unit. Then the nextion display is attached to the ESP 32 development board. Figure 6.5 shows the arrangement of the overall components. The entire system is powered by a 5V power supply.

6.3 Weather Prediction Algorithm

Zambretti algorithm is used for weather prediction in this study. First, using the code mentioned below, current atmospheric pressure is converted to sea level pressure.

```
sea_Level_Pressure = pressure * pow(1 - (0.0065 * altitude_meters) temperature +
(0.0065 * altitude_meters) + 273.15), -5.257 );
```

then it will be stored in an array. For that, storing an array has to be executed every 10 minutes. Therefore, using the millis() function, the below code was used to call the data saving function every 10 minutes. Since it runs in the loop function, no programme delay will happen due to that.

```
timer = millis();
 if (timer - timer_Save_Data >= 600000) {
   save_Data();
   timer_Save_Data = timer;
 }
```

Then the below function (Figure 6.6) will be called and based on the counter value, the related index of the array will save the sea level pressure on it. Then using the first 5 indexes and the last 5 indexes, the trend of the atmospheric pressure will be calculated.

```
void save_Data() {
  if (counter == 18) {
    counter = 0;
  } else {
    pressureArray[counter] = sea_Level_Pressure;
    counter++;
  }

  current_Pressure = (pressureArray[17] + pressureArray[16] + pressureArray[15] +
pressureArray[14] + pressureArray[13]) / 5;
  previous_Pressure = (pressureArray[0] + pressureArray[1] + pressureArray[2] + pressureArray[3]
+ pressureArray[4]) / 5;

  if ((current_Pressure - previous_Pressure) >= 1.6 && sea_Level_Pressure >= 947 &&
sea_Level_Pressure <= 1030) {
    trend = "Rising";
  } else {
    if ((previous_Pressure - current_Pressure) >= 1.6 && sea_Level_Pressure >= 960 &&
sea_Level_Pressure <= 1033) {
      trend = "Falling";
    }  else {
      if ((current_Pressure - previous_Pressure) <= 1.6 || (previous_Pressure - current_Pressure)
<= 1.6 && sea_Level_Pressure >= 947 && sea_Level_Pressure <= 1030) {
        trend = "Steady";
      } else {
        trend = "Error";
      }
    }
  }

  get_Zambretti_Value();
}
```

Figure 6.6: Save Data Function

Then, it will be checked that the pressure trend is rising or falling by 1.6 hPa or keep it less than 1.6 hPa increment or decrement. Also, there will be specific ranges for the pressure trend that only considered required values only. If the trend is falling, SLP has

to be within 985 hPa and 1050 hPa. Also, the SLP has to be dropped by 1.6 hPa in 3 hours. . If the trend is steady, SLP has to be within 960 hPa and 1033 hPa. Also, the SLP has to be no drop or no rise by 1.6 hPa in 3 hours. . If the trend is rising, SLP has to be within 947 hPa and 1030 hPa. Also, the SLP has to be raised by 1.6 hPa in 3 hours. If none of the above happened, that function will return an error status. Then it called another function to get the Z value.

```
void get_Zambretti_Value() {
  if (trend == "Falling") {
    zambretti_Calc = 127 - 0.12 * sea_Level_Pressure;
  } else {
    if (trend == "Steady") {
      zambretti_Calc = 144 - 0.13 * sea_Level_Pressure;
    } else {
      if (trend == "Rising") {
        zambretti_Calc = 185 - 0.16 * sea_Level_Pressure;
      } else {
        weather_Prediction_Status = weather[0];
      }
    }
  }
  weather_Prediction = round(zambretti_Calc);
  add_Wind_Effect();
}
```

Figure 6.7: Function for getting the Zambretti Value

Figure 6.7 shows that function and based on the SLP trend, will use separate formulas to calculate the Z value. If the trend is falling, $Z = 127 - 0.12 \times$ Sea Level Pressure. If the trend is steady, $Z = 144 - 0.13 \times$ Sea Level Pressure. And, if the trend is rising, $Z = 185 - 0.16 \times$ Sea Level Pressure. If the save data function returns an error status, this will return the error status stored in the weather[] array at index 0. Then the value will be round and it will convert to the Z value as an integer. Then using another function, the effect of the wind is matched to the Z value.

Then wind effect will be added to the Z value using the add_Wind_Effect() function (Figure 6.8). if the wind is coming from East or West, 1 will be added to the Z value. If the wind is coming from the South, 2 will be added to the Z value. Or else, if the wind is coming from the North, nothing will be added t the Z value. Then finally, using the weather[] array and the Z value, the correct prediction will be identified.

```
void add_Wind_Effect() {
  if (wind_Angle >= 46.0 && wind_Angle <= 135.0) {
    weather_Prediction = weather_Prediction + 1;
  } else {
    if (wind_Angle >= 136.0 && wind_Angle <= 225.0) {
      weather_Prediction = weather_Prediction + 2;
    } else {
      if (wind_Angle >= 226.0 && wind_Angle <= 315.0) {
        weather_Prediction = weather_Prediction + 1;
      } else {
        weather_Prediction = weather_Prediction + 0;
      }
    }
  }
  weather_Prediction_Status = weather[weather_Prediction];
}
```

Figure 6.8: Function for Add the Wind Effect to the Z Value

6.4 Calculate Minimum, Maximum and Average

Figure 6.9 shows the function for calculating the max, min and average of a single weather parameter. First, the counter for average values will be updated and added 1 at a time. Then the temperature value is mapped to display the temperature gauge in the Nextion display. 0 and 50 determine the minimum and maximum possible values for the relevant variable. Then, 17 and 88 determine the minimum and maximum values that we need to output after the mapping is done. These values were identified by using the Nextion Editor. Those are the min and max values that are used to display the min and max of the related gauge.

Then, in the setup function maximum and average values are equalled to 0. And minimum value is equalled to som higher value which normally never occurred in practice. Then, the current temperature will be checked with the minimum temperature value whether is it less than or equal to the minimum temperature. If yes, the current temperature will be allocated to the minimum temperature. For maximum temperature, it will be checked if the current temperature is greater than or equal to the maximum temperature, if yes, the current temperature will be allocated to the maximum temperature.

```
void calculateDetailedWeatherData() {

  avg_Counter++;

  temperature_Mapped = map(temperature, 0, 50, 17, 88);

  //Minimum Temperature
  if (temperature <= temperature_Min ) {
    temperature_Min = temperature;
  } else {
    temperature_Min = temperature_Min;
  }

  temperature_Min_Mapped = map(temperature_Min, 0, 50, 17, 88);

  //Average Temperature
  temperature_Total = temperature_Total + temperature;
  temperature_Avg = temperature_Total / avg_Counter;

  temperature_Avg_Mapped = map(temperature_Avg, 0, 50, 17, 88);

  //Maximum Temperature
  if (temperature >= temperature_Max) {
    temperature_Max = temperature;
  } else {
    temperature_Max = temperature_Max;
  }

  temperature_Max_Mapped = map(temperature_Max, 0, 50, 17, 88);
```

Figure 6.9: Function for Calculate Minimum, Maximum and Average

For average temperature, the current temperature will be added to the total temperature. Then it will be divided using the average counter and the average temperature will be derived. At the end of each calculation, relevant data will be mapped to the related gauge's values in the Nextion display.

6.5 User Interfaces

User interfaces play an important role in this study. All the captured and calculated data are shown using these user interfaces. When the system is loading, the home page (Figure 6.10) will be loaded and, it contains all the basic weather data. As an initial component, the date and the time were shown in the top left corner. In the same line, there are 3 tabs for accessing the current weather data, detailed weather data and the settings of the system. In current weather data, temperature, humidity, atmospheric pressure, wind direction in text and the angle in degrees, wind speed, rainfall, dew point, heat index and the moon phase will be displayed. Also, since there are 2 pages for

Figure 6.10: Home Page

current weather data, there are 2 navigation arrows displayed on the mid-left and the mid-right. Also, to feel the time of the day, the background of the home page will be changing during the day automatically. There are 8 backgrounds available to change throughout the day for more interactiveness.

Figure 6.11: Home Page 2

When navigating using the left or right arrows, the second home page (Figure 6.11) will be loaded. This page includes the date and the time with 3 tabs to access the features of

the UI. This page includes sunrise time, sunset time, solar noon time and weather prediction. Also, to feel the lapse of time, this page contains automatic background changing features identical to the home page. The navigation buttons help to navigate to the home page again.

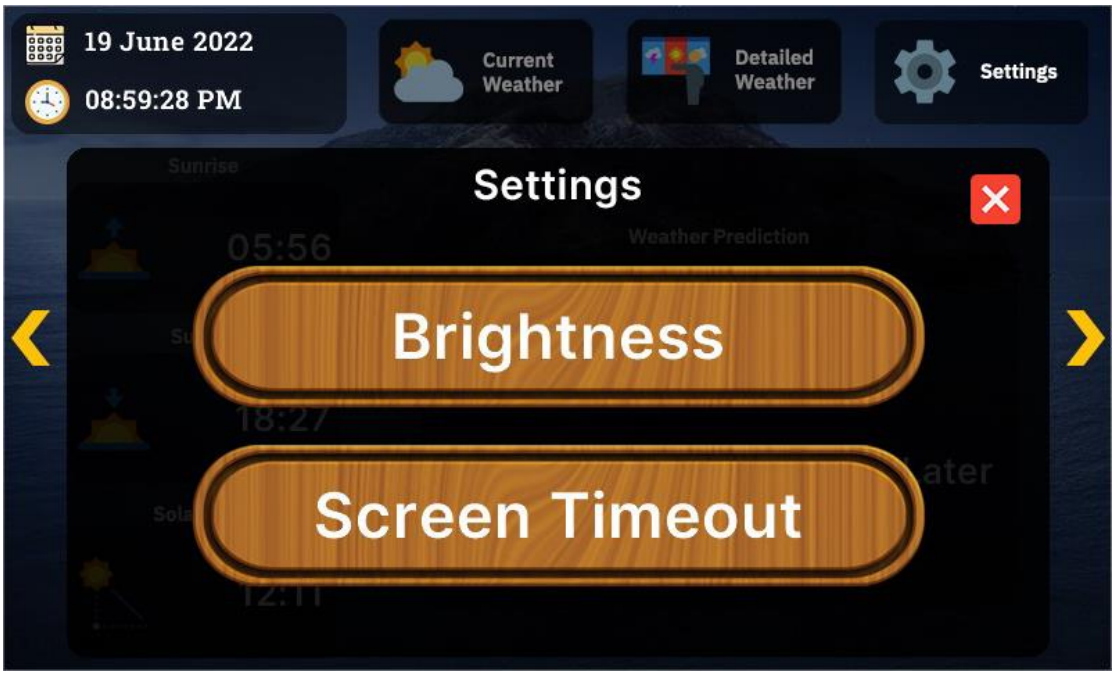


Figure 6.12: Settings Home Page

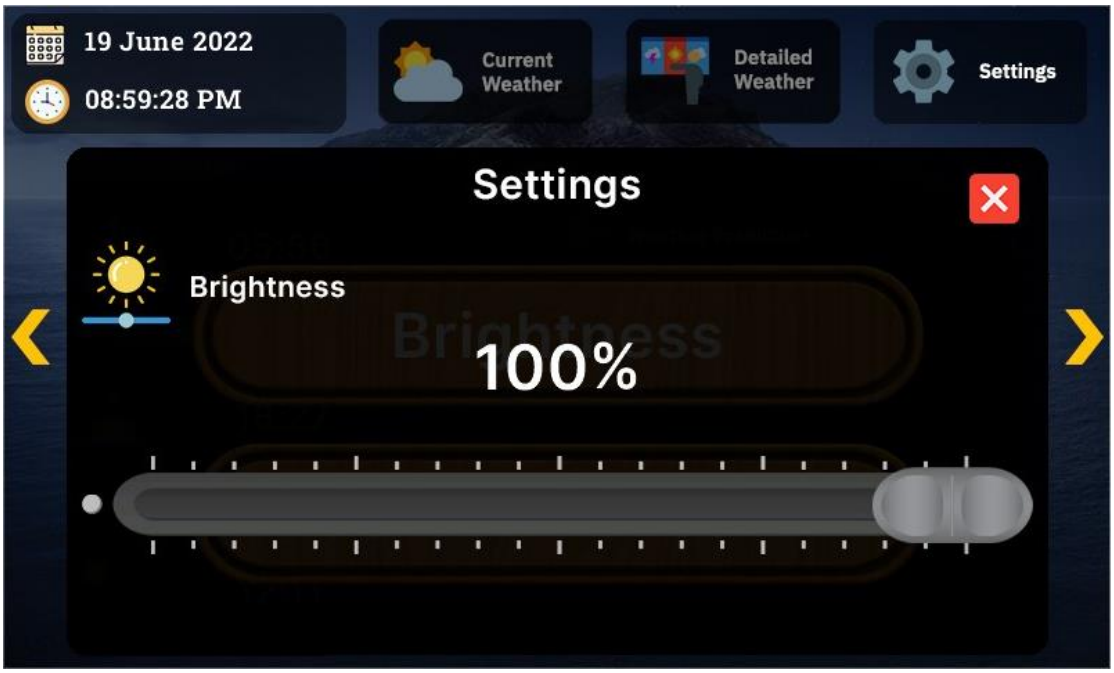


Figure 6.13: Brightness Settings Page

When touching the settings tab, it will navigate to the settings home page (Figure 6.12). This page includes 2 main settings. One is for adjusting the brightness and the other

one is for adjusting the screen timeout. These settings are introduced to save the life span of the display.

When touching the Brightness button, it will navigate to the brightness settings page (Figure 6.13). adjusting the brightness is a feature of the display. Therefore, there is no coding done from the microcontroller to adjust the brightness. First, a global variable for brightness value has to be declared in the “program.s” window (Figure 6.14). Then for the slider, the touch release event and the touch move events, the code that is shown in Figure 6.15 has to be entered. Here, “dim” is a command that is used to set the brightness value of the display. When it is changed, it can be maintained throughout the pages that are loaded.

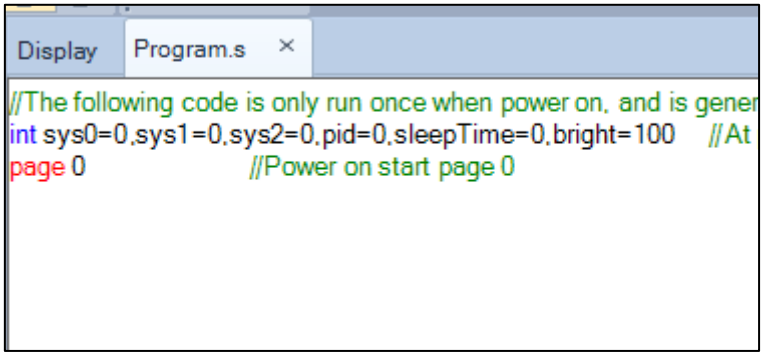


Figure 6.14: Variable declaration for Brightness

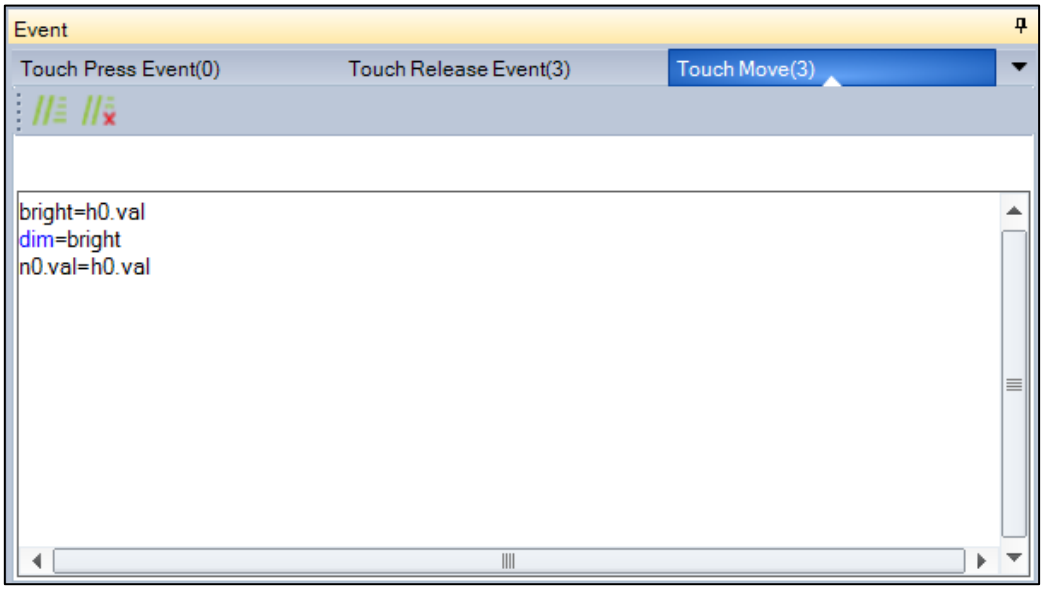


Figure 6.15: Brightness Changing Code

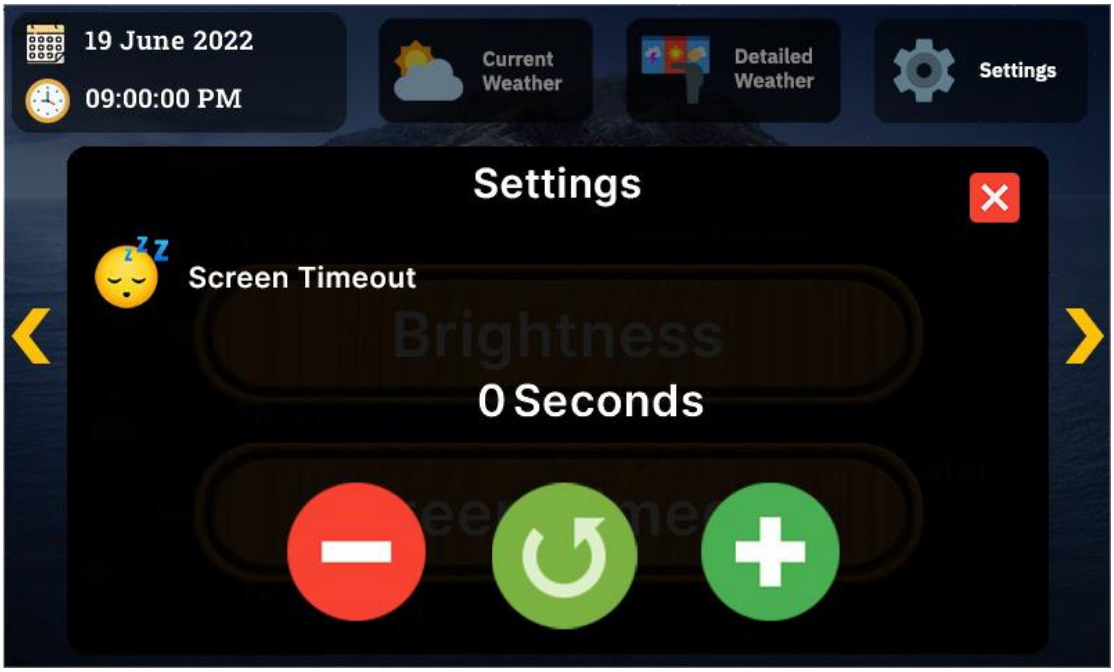


Figure 6.16: Screen Timeout Settings Page

Figure 6.16 shows the screen timeout settings page. Here, there are 3 buttons to increase, decrease and reset the screen timeout. To set the screen timeout, first, a variable has to be declared as per Figure 6.14. Then to increase and decrease time, the code shown in Figure 6.17 has to be entered in the touch release event. Then, the below

```
if(n0.val<=64985)
{
  n0.val=n0.val+15
}else
{
  n0.val=n0.val
}
sleepTime=n0.val
```

```
if(n0.val>=15)
{
  n0.val=n0.val-15
}else
{
  n0.val=n0.val
}
sleepTime=n0.val
```

Figure 6.17: Screen Timeout Increase and Decrease Code

code has to be entered on the home page under the pre-initialise event defined in Figure 6.14.

```
thsp=sleepTime
thup=1
```

Here, “thsp” determines the “Sleep on No Touch” value. It can be a minimum of up to 3 seconds and a maximum of up to 65535 seconds. Also, “thup” determines the “Auto Wake on Touch”. If this value is 0, even touch response is detected, the display will not wake up. If this value is 1, after a touch response is detected, the display will wake up.

The reset button will reset the sleep time value. By pressing the close button on the settings page, will navigate to the last home page that was loaded before.

By touching the detailed weather tab, it will navigate to the detailed weather page. Initially, it will be landed to a detailed view of the temperature (Figure 6.18). This page includes the date and the time in the top left corner. Hereafter, every detailed weather

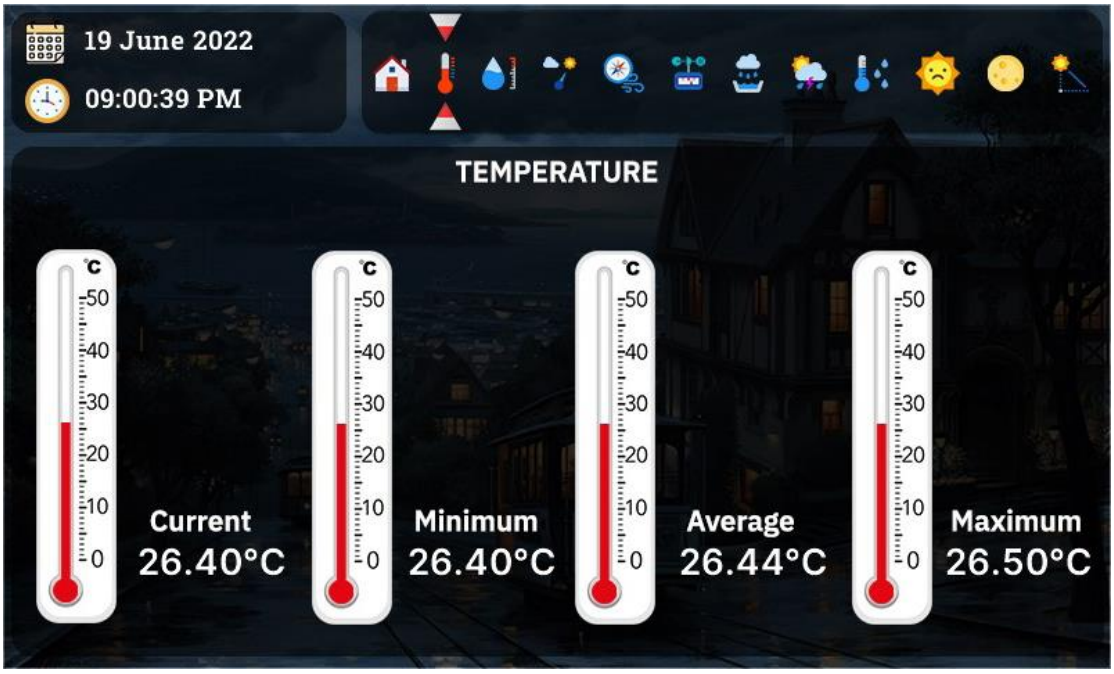


Figure 6.18: Detailed Weather - Temperature

page includes the date and the time. Then, there is a navigation panel that can jump any detailed view of weather data. To maintain consistency, the same icons that were used on home page 1 and home page 2 are used here. The selected page can be identified by the 2 triangles pointing toward the related icon. In the detailed area, current, minimum, average and maximum temperature values are displayed. Also, the gauges are indicating the corresponding values to it. All the gauges are shown in detail weather pages are fully functional and it would give a more interactive look and a comparative view to the user.

Figure 6.19 shows a detailed view of humidity. When paying attention to the gauges, those showing the exact values are precisely based on the corresponding parameters. When the date is changed, those minimum, maximum and average values are getting reset. Therefore, these values are valid for an existing day only.

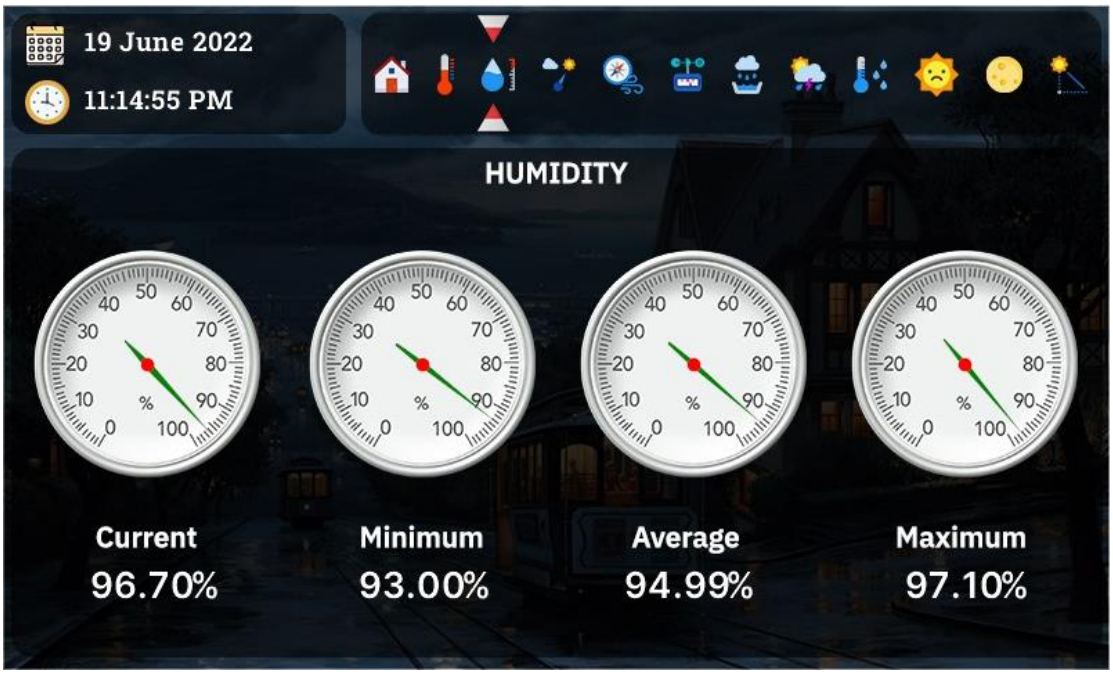


Figure 6.19: Detailed Weather - Humidity

Figure 6.20 shows the atmospheric pressure data. Here also, minimum, maximum, average and current values are displayed using numbers and gauges. Since the range is quite high, the difference between minimum, maximum and average is difficult to see on this page.

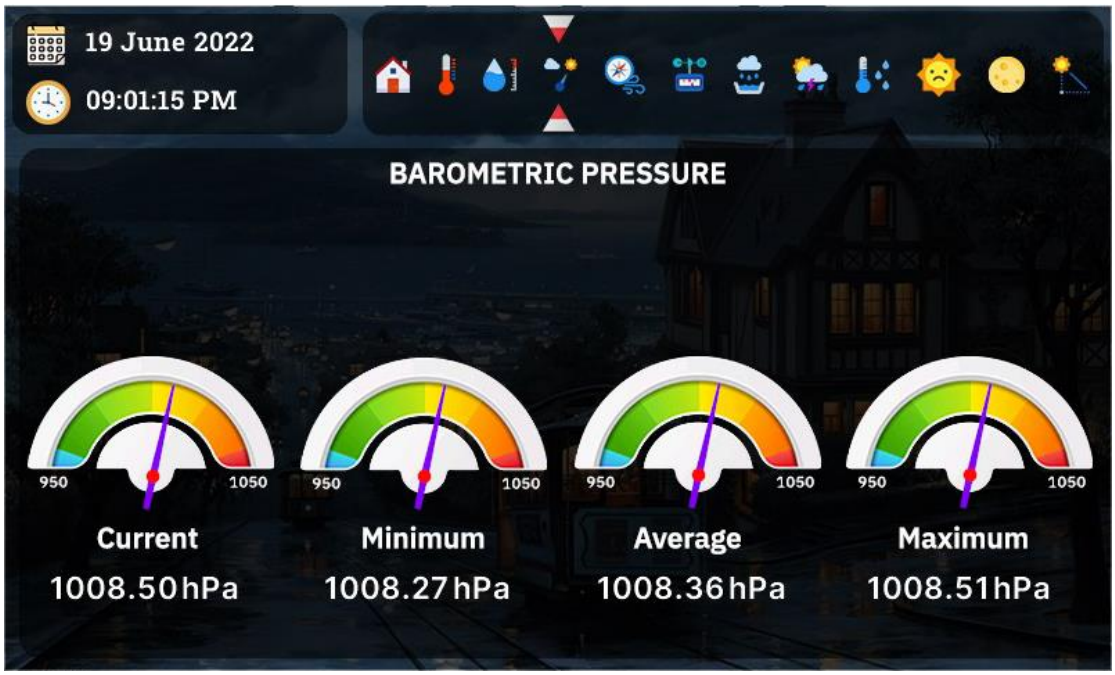


Figure 6.20: Detailed Weather – Atmospheric Pressure

Figure 6.21: Detailed Weather – Wind Direction

Figure 6.21 shows the detailed page on wind direction. This page includes wind direction in text and the degrees. Since the direction texts are too long, abbreviations of those directions are used. Also, there is an icon showing the wind direction in the text and pointing towards that direction for better understanding. But this pointing direction of the arrow is generated assuming that pointing to the top of the display is considered as North.

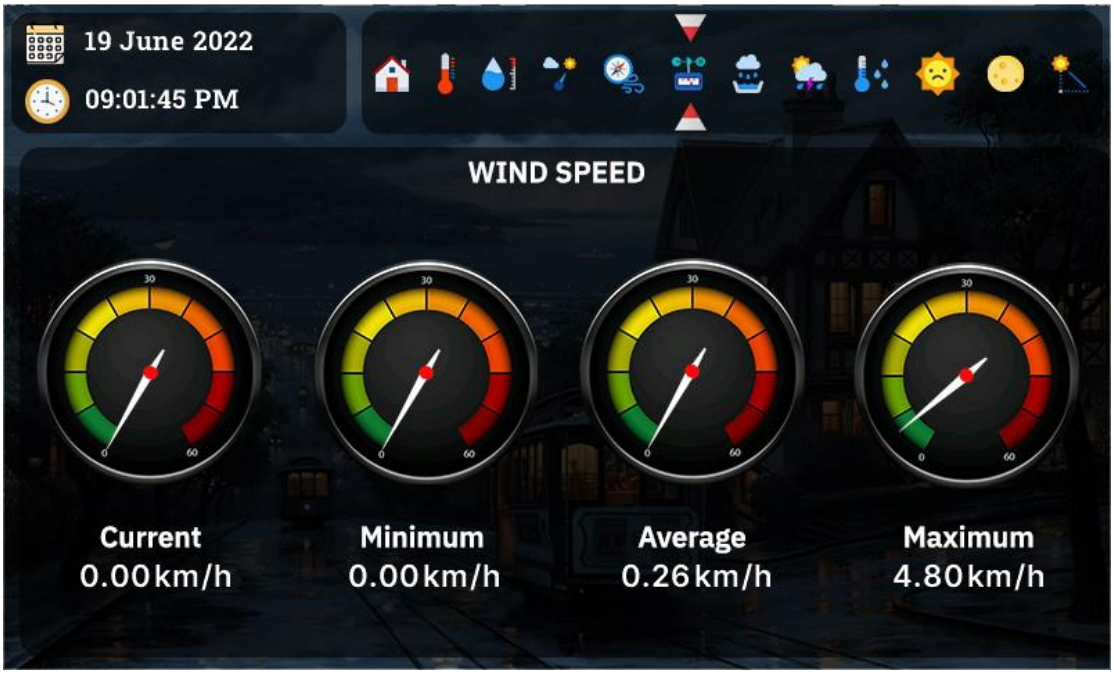


Figure 6.22: Detailed Weather – Wind Speed

Figure 6.22 shows the wind speed detailed data. Since there can be 0 wind speed situations, almost all the time minimum wind speed will be 0.00 km/h. Figure 6.23

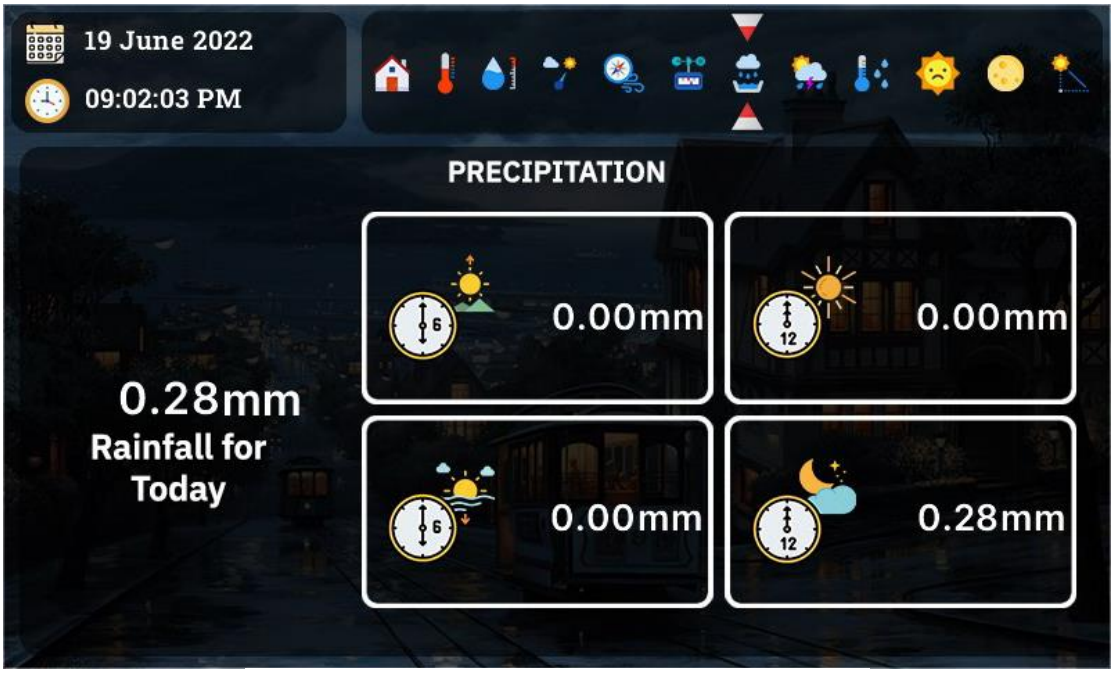


Figure 6.23: Detailed Weather – Precipitation

shows the precipitation details. Since it is difficult to measure the rainfall as a minimum, maximum and average, it was divided into 4 parts. Those parts have 6 hours and the top left part shows the rainfall from midnight to 06:00 am. The top right part shows the rainfall from 06:01 am to 12:00 noon. The bottom left part shows the rainfall from 12:01 pm to 06:00 pm. The bottom right part contains the rainfall from 06:01 pm to midnight. Those values are calculated based on the rainfall coming through the outdoor unit and using the hour value, those will be partitioned. This will help to analyse which quarter that the rainfall has happened mostly etc.

Figure 6.24 shows the weather prediction detailed view. This contains the weather prediction in text and there is a graphic representation of that weather prediction. Based on the prediction, there are 24 graphics used to represent the 26 weather predictions. Since it required at least 3 hours to predict the weather accurately. Therefore, for the first 10 minutes, it will show an unknown weather prediction icon. Since the texts are difficult to understand, graphical representation of those is very helpful to maintain the user interaction.

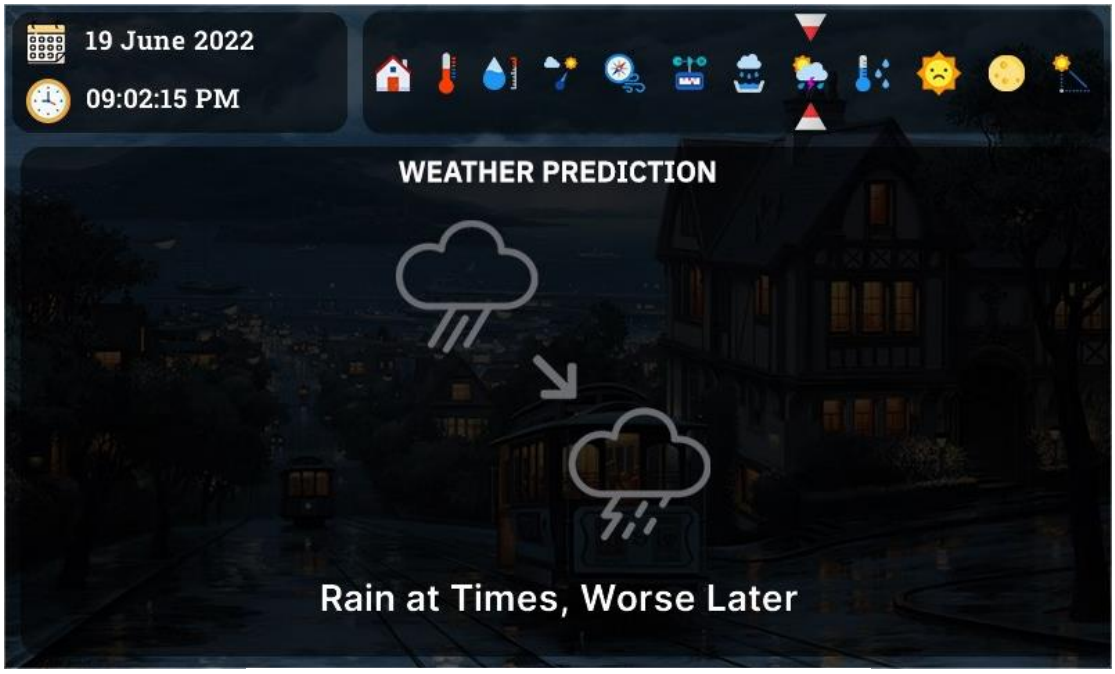


Figure 6.24: Detailed Weather – Weather Prediction

Figure 6.25 shows the detailed view of the dew point. Dew point temperature is “the temperature the air needs to be cooled to (at constant pressure) to achieve a relative humidity (RH) of 100%” [59]. Then water droplets become to form dew. As previous temperature measurements, this has the same structure that displayed maximum, minimum and the average dew point other than the current reading.

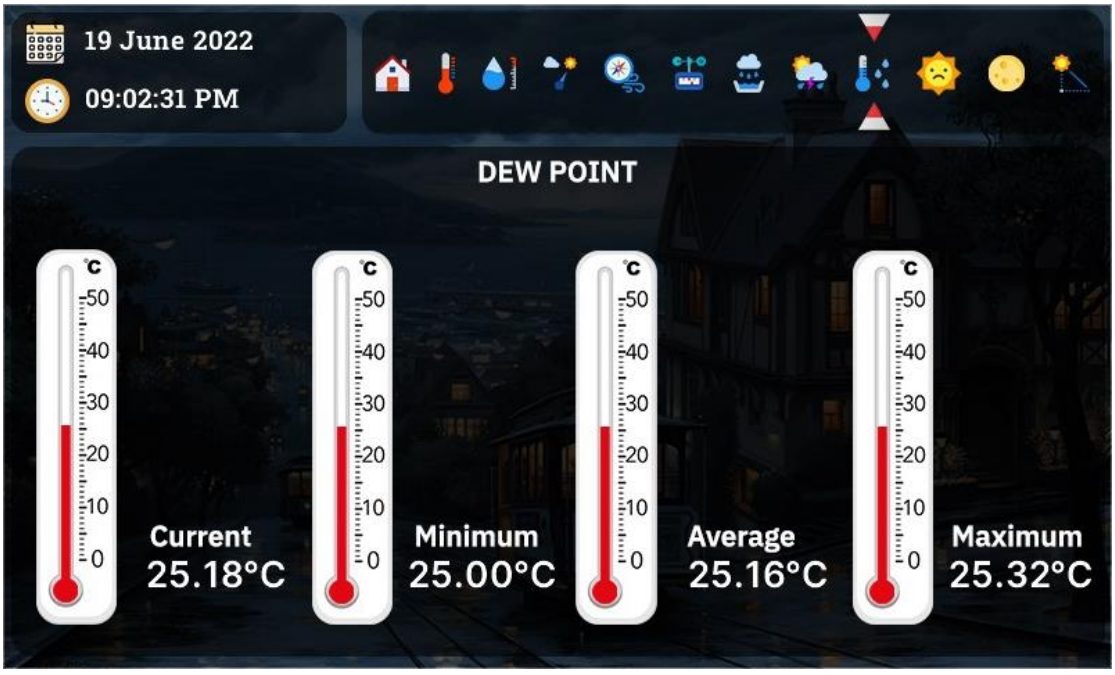


Figure 6.25: Detailed Weather – Dew Point

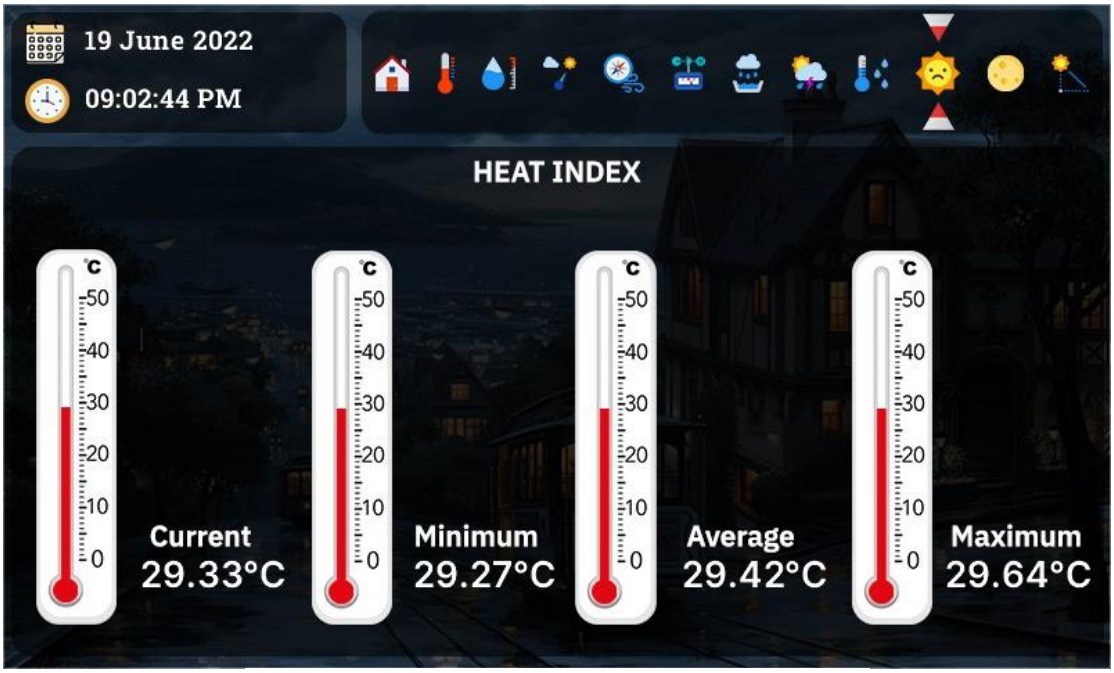


Figure 6.26: Detailed Weather – Heat Index

Figure 6.26 shows the heat index data. The heat index is “the temperature that feels in a human body when relative humidity is combined with the air temperature” [60]. The human body dissipates the heat using by perspiring or sweating. If the humidity level is high in the surrounding, it is difficult to evaporate the sweat. Therefore, the human body will feel additional heat than the actual heat. Extreme value of heat index can be harmful to the human body and can be caused a heat stroke too.

Figure 6.27: Detailed Weather – Moon Phase

Figure 6.27 shows the moon phase of the current day. This page includes the moon phase in text and an interactive sample graphic of the moon phase. Since the graphics are involved, it is easy to understand what the moon phase is.

Figure 6.28: Detailed Weather – Solar Cycle

Figure 6.28 shows the solar cycle of the day. First, it shows the sunrise time. Then, in the middle, it shows the solar noon which is at what time the sun will be directly above us. Finally, it shows the sunset time. The path of the sun is shown in yellow colour to understand the graphic easily.

6.6 Cost of the Project

Since this study has physical development, some of the components have to purchase newly and most are available during the implementation period. Therefore, the cost for all the components is shown below in Sri Lankan Rupees (LKR).

Table 6.1: Cost of Study

Description	Cost (LKR)
Arduino Mega 2560 Development Board	2,450.00
ESP 32 Development Board	950.00
Nextion 5-inch Display (with enclosure)	12,000.00
Weather data capturing unit	8,000.00
Buck Converter (20A)	600.00

Boost Converter (1A)	150.00
TP4056 Module (2 units)	120.00
18650 Batteries (3200 mAh)	680.00
18650 Batter Holder (2 units)	120.00
nRF24L01 PA LNA module with Power Supply Module (2 units)	970.00
RTC Module	260.00
20W Solar Panels (2 units)	5,600.00
ESP 32 Expansion Board (2 units)	650.00
AM2301 module	650.00
MPL3115A2 module	750.00
Rain, wind direction and speed capturing support unit	375.00
Plastic Project Box - Large	850.00
Plastic Project Box - Small	375.00
Wires and other components	450.00
Total Cost	**36,000.00**

Based on the above calculations, the total cost of the project was LKR 36,000.00.

6.7 Power Consumption

Microcontroller-based development boards are less power-consuming devices. Therefore, this project consumes less power. When measuring the power consumption, below mentioned facts could be able to identify. The outdoor unit utilises 135 milliamperes (mA) as average current consumption (Figure 6.29 part a). Since it used 5V, the average power consumption was 0.675 Watts (W).

Figure 6.29: Current consumption of (a) Outdoor Unit (b) Indoor Unit with Display On (c) Indoor Unit with Display on Sleep Mode

Considering the Indoor units consume 480 mA of average current when the complete unit is functioning at its maximum capacity. When this reading was extracted, the indoor units were connected to the Wi-Fi for capture time and syncing the data to the Adafruit IO IoT cloud and the display runs at maximum brightness without sleep mode functioning. Since this unit also runs on 5V, this consumes 2.4W of average power consumption. when the display is in sleep mode, it consumes 259 mA of average current and that consumes 1.295W of average power consumption.

Table 6.2: Total Power Consumption

Description	**Outdoor Unit**	**Indoor Unit (Display Sleep Mode Off)**	**Indoor Unit (Display Sleep Mode On)**
Running Voltage (V)	5	5	5
Average Current Consumption (mA)	135	480	259
Average Power Consumption (W)	0.68	2.40	1.30
Average Power Consumption for Day (kWh)	0.02	0.06	0.03
Average Power Consumption for Month (kWh)	0.49	1.73	0.93
Average Power Consumption for Year (kWh)	5.92	21.04	11.35

Table 6.2 shows the average power consumption of the outdoor unit and indoor unit with the display on and off. Based on that table, the outdoor unit consumes an average of 5.92 units of electricity per year. Also, the indoor unit consumes an average of 21 units of electricity per year when the display is on and runs at 100% of brightness. If the display runs in sleep mode, it will consume an average of 11 units of electricity per year.

Therefore, it can be identified that this unit consumes less power and it would be a suitable solution for a weather prediction system with low power consumption.

Chapter 7 - Evaluation

7.1 Background for the Evaluation

This chapter includes the evaluation of the data captured using the weather prediction system. History data was stored in the Adafruit IO IoT cloud service and based on the used plan, it was able to keep the history data up to 30 days. Therefore, it would be able to extract the history data from 06th July 2022 to 05th August 2022. Hence, there are 30 days of historical data to analyse. The data were analysed using Microsoft Excel. Pivot Tables, Pivot Graphs and other custom Excel formulas are used to analyse those data.

7.2 Temperature

Among the 30 days of historical data, initially, data was used to identify the average temperature for each day. Based on the analysed data, Figure 7.1 would be able to

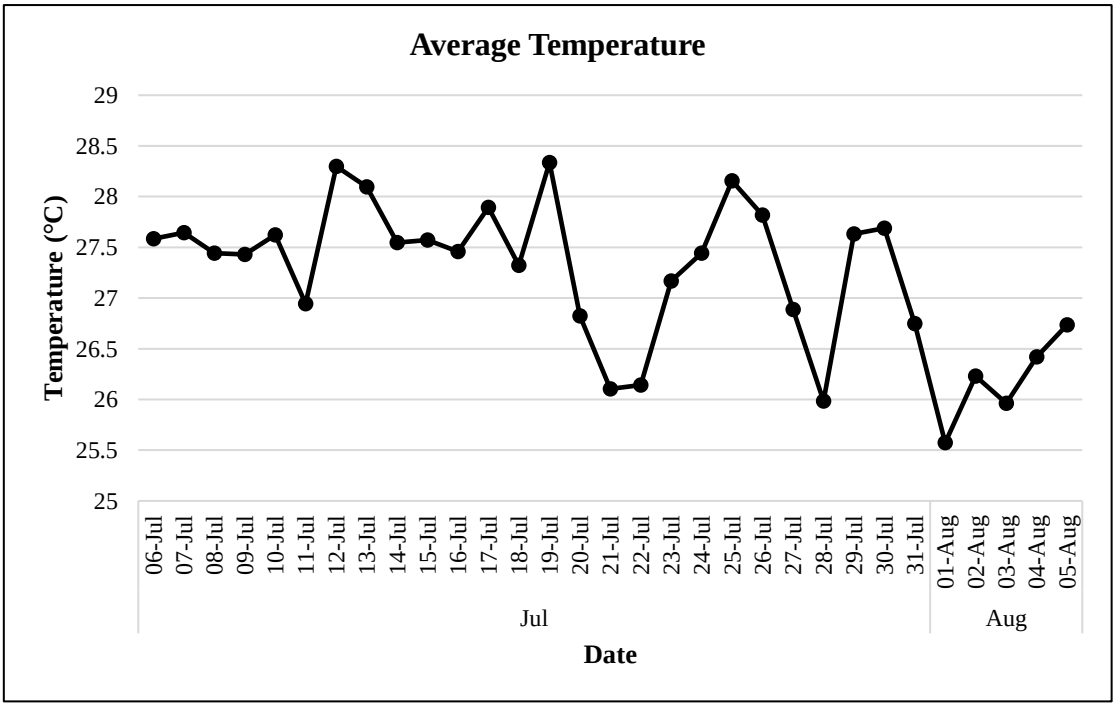


Figure 7.1: Average Temperature

generate. Based on that it would be able to identify that the average temperature varied from 25 °C to 28 °C. Also, based on the temperature distribution shown in Figure 7.2, it would be able to identify that from the entire temperature range, most of the recorded temperatures are from 25 °C to 28 °C.

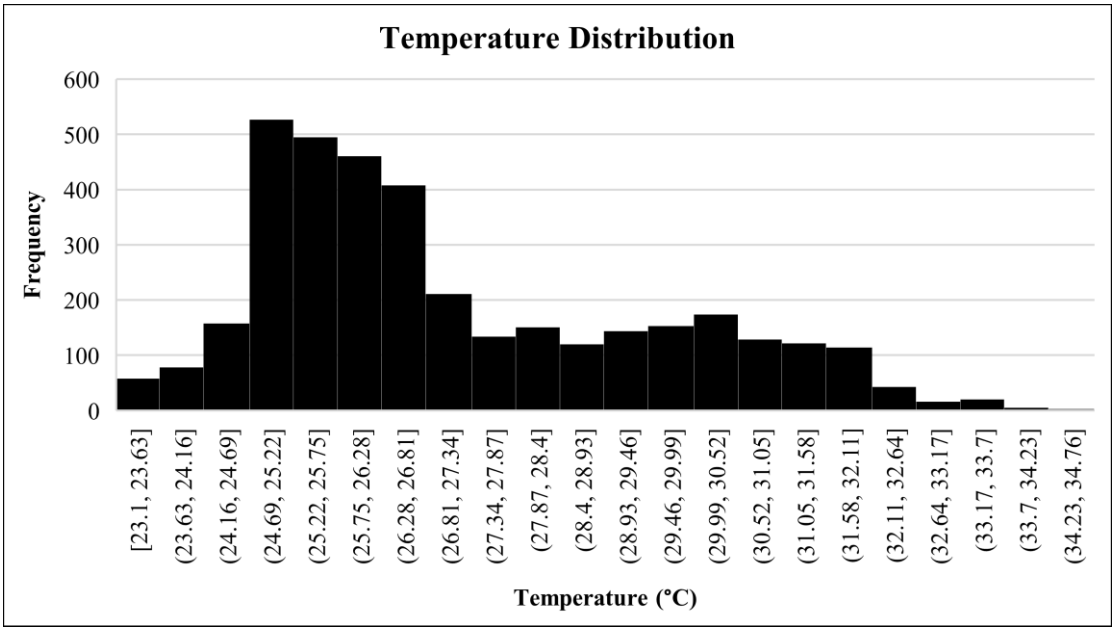


Figure 7.2: Temperature Distribution

Considering the maximum and minimum temperature values recorded within the day, Figure 7.3 generated and based on that, it would be able to identify the highest temperature was recorded on 27th July which is 34.7 °C and the lowest temperature was recorded on 28th July which is 23.1 °C.

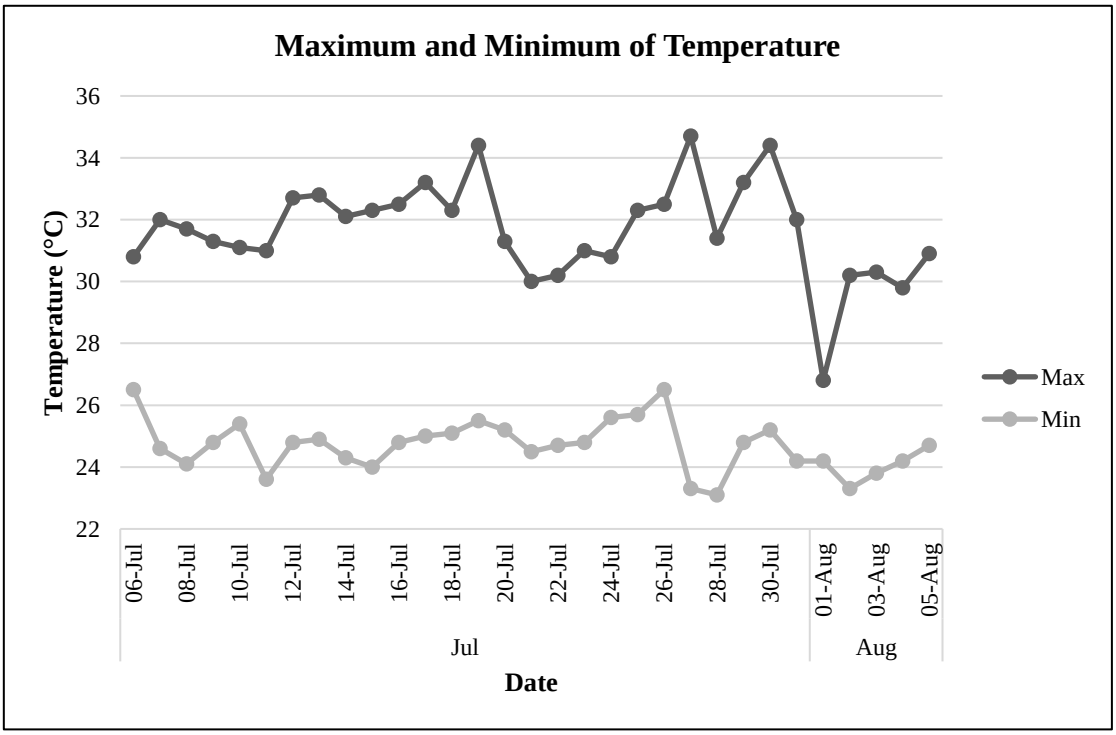


Figure 7.3: Maximum and Minimum Temperature

Considering the highest and the lowest temperature recorded time for 30 days, Figures 7.4 and 7.5 was generated.

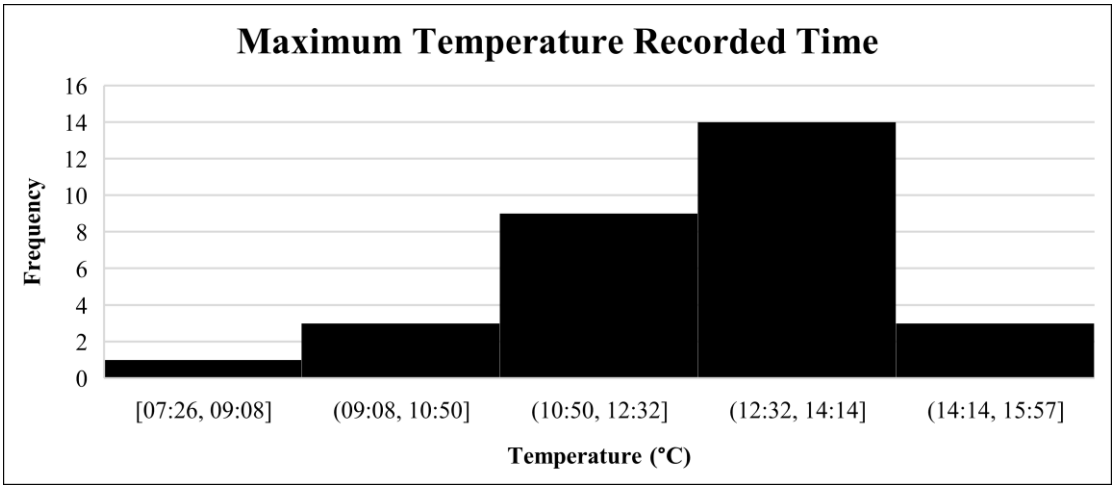


Figure 7.4: Maximum Temperature Recorded Time

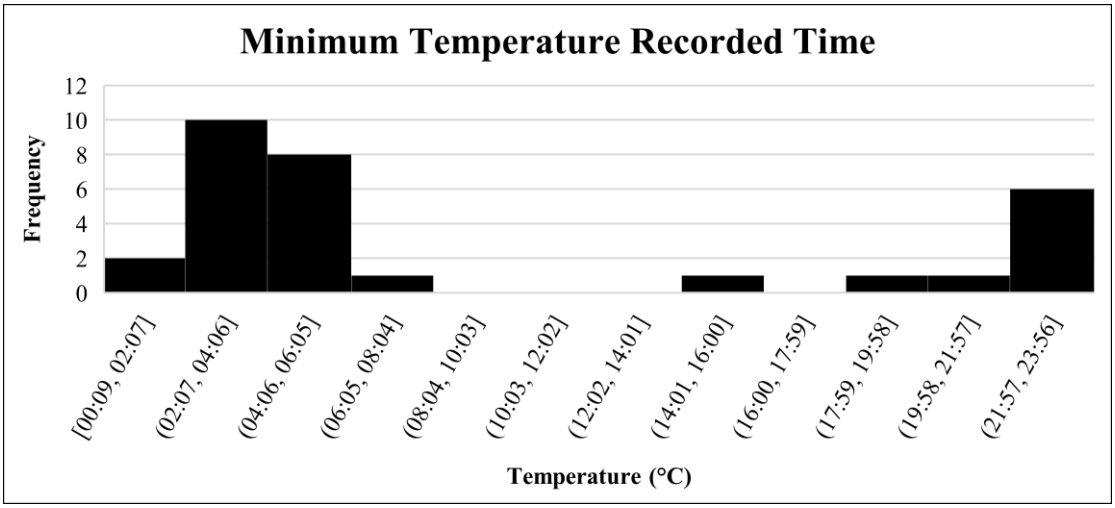


Figure 7.5: Maximum Temperature Recorded Time

Based on that, it would be able to identify that, the maximum temperature will be recorded approximately from 10:30 am to 02:30 pm. The minimum temperature will be recorded approximately from 09:30 pm to 06:00 am.

7.3 Humidity

Using those 30 days of historical data, the average humidity level for each day was calculated. Figure 7.6 shows how the average humidity will be changed each day. Based on that it would be able to identify that the average humidity varied from 81% to 95%. Also, based on the humidity distribution shown in Figure 7.7, it would be able to identify that from the entire humidity range, most of the recorded humidity values are from 92% to 100%.

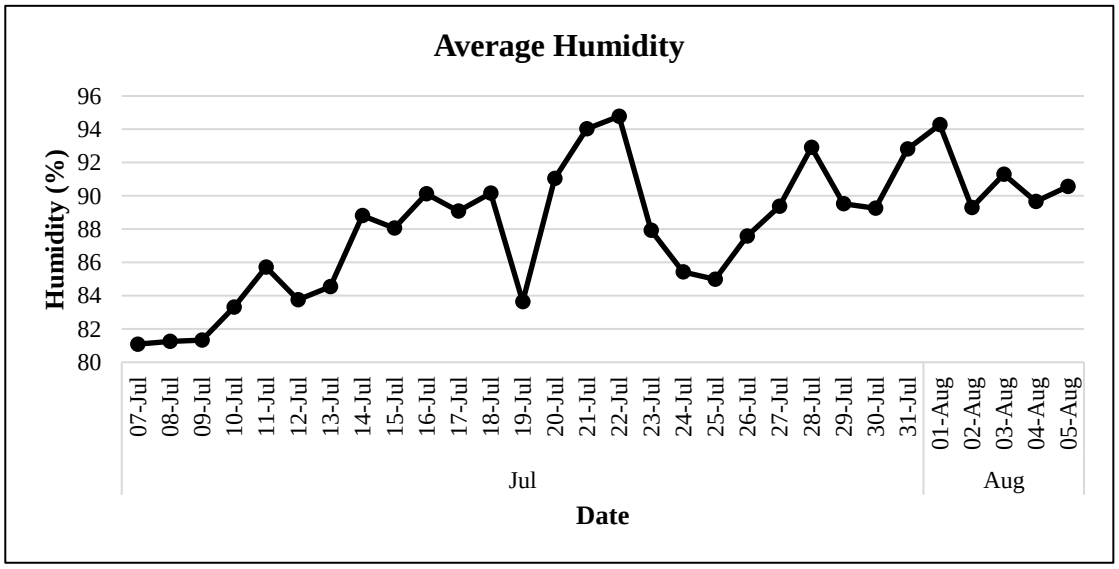


Figure 7.6: Average Humidity

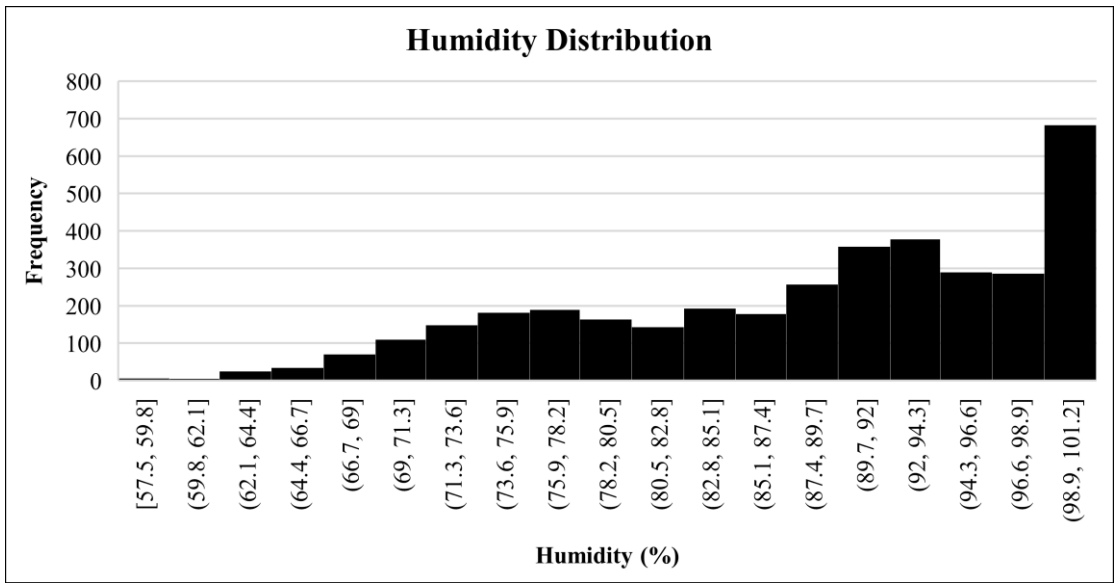


Figure 7.7: Humidity Distribution

Considering the maximum and minimum humidity values recorded within the day, Figure 7.8 generated and based on that, it would be able to identify the highest humidity was recorded on multiple days which is 99.9% and the lowest humidity was recorded on 30th July which is 57.5%.

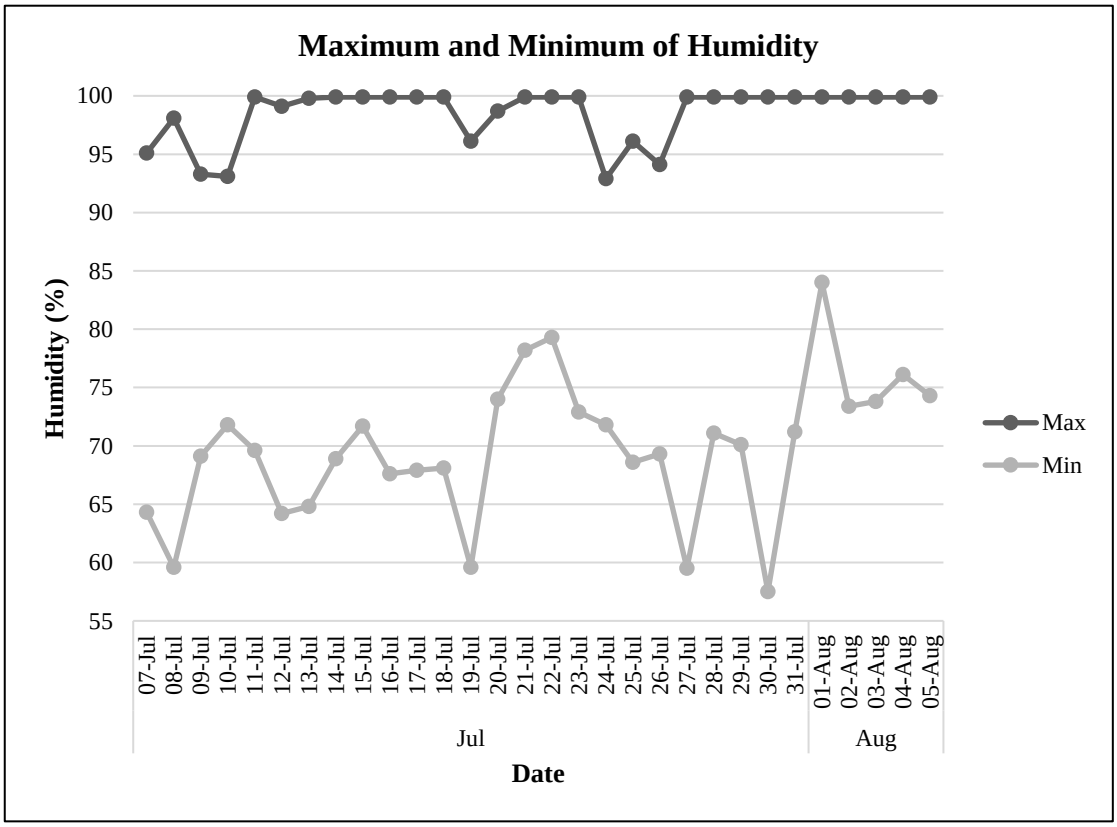


Figure 7.8: Maximum and Minimum Temperature

Considering the highest and the lowest humidity recorded time for 30 days, Figures 7.9 and 7.10 was generated. Based on that, it would be able to identify that, the maximum humidity will be recorded approximately from 12:00 am to 09:30 am. The minimum humidity will be recorded approximately from 12:30 pm to 03:30 pm.

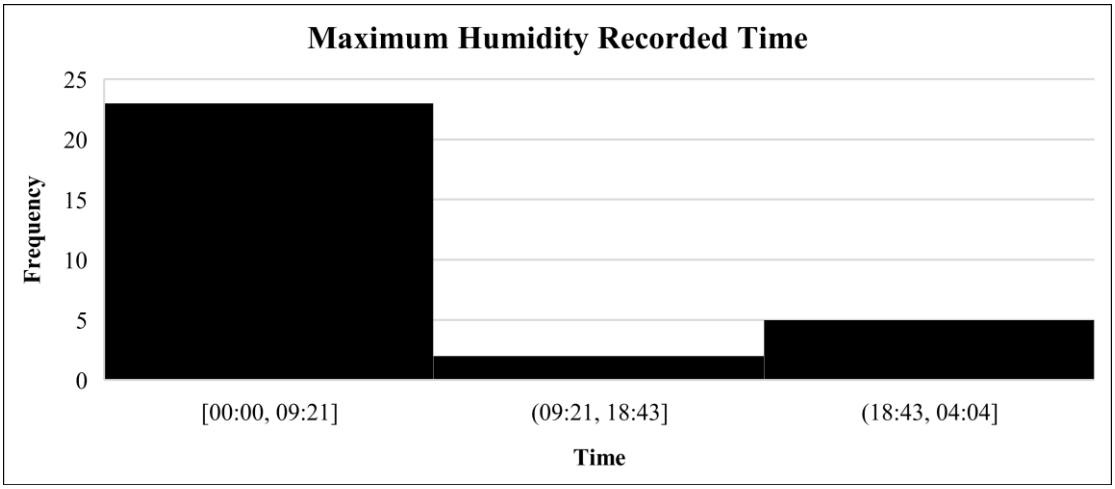


Figure 7.9: Maximum Humidity Recorded Time

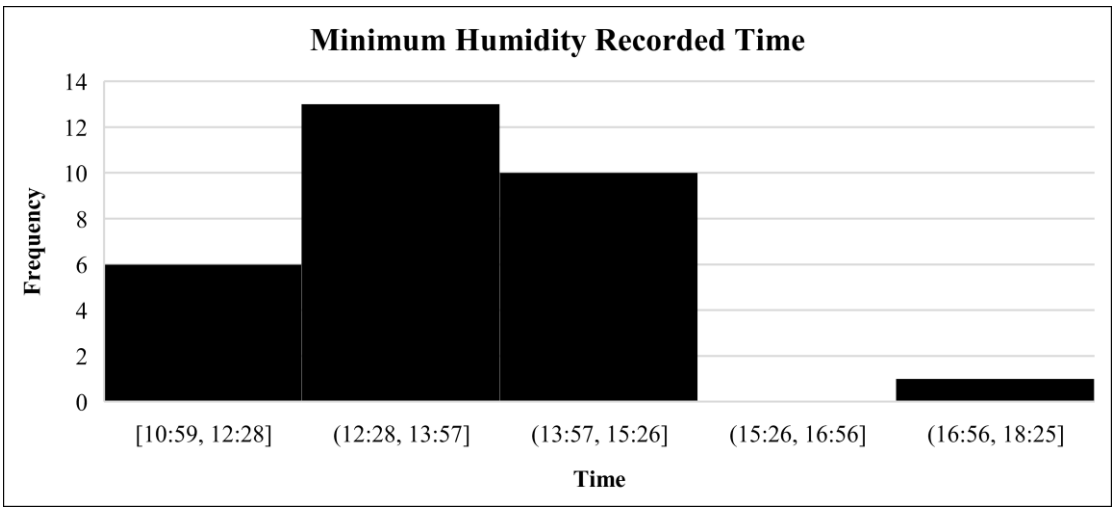


Figure 7.10: Minimum Humidity Recorded Time

7.4 Atmospheric Pressure

Using those 30 days of historical data, the average atmospheric pressure level for each day was calculated. Figure 7.11 shows how the average atmospheric pressure will be changed each day. Based on that it would be able to identify that the average atmospheric pressure varied from 1006 hectopascals (hPa) to 1011.5 hPa. Also, based on the atmospheric pressure distribution shown in Figure 7.12, it would be able to identify that from the entire atmospheric pressure range, most of the recorded atmospheric pressure values are from 1007 hPa to 1008 hPa.

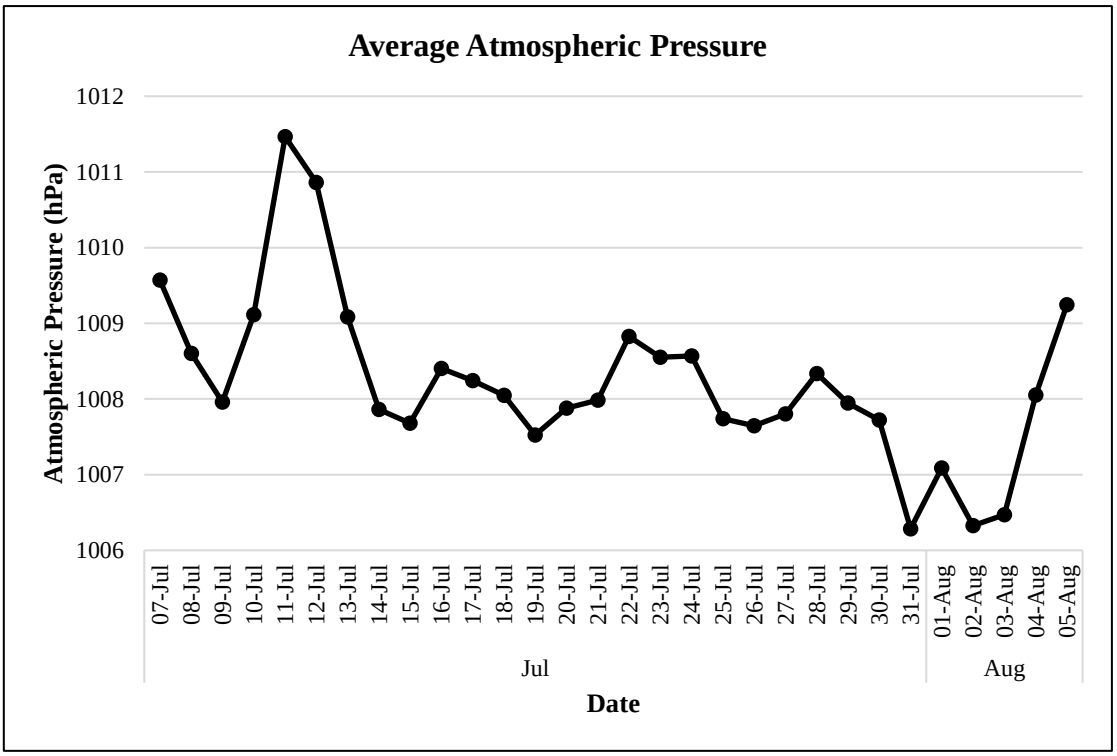


Figure 7.11: Average Atmospheric Pressure

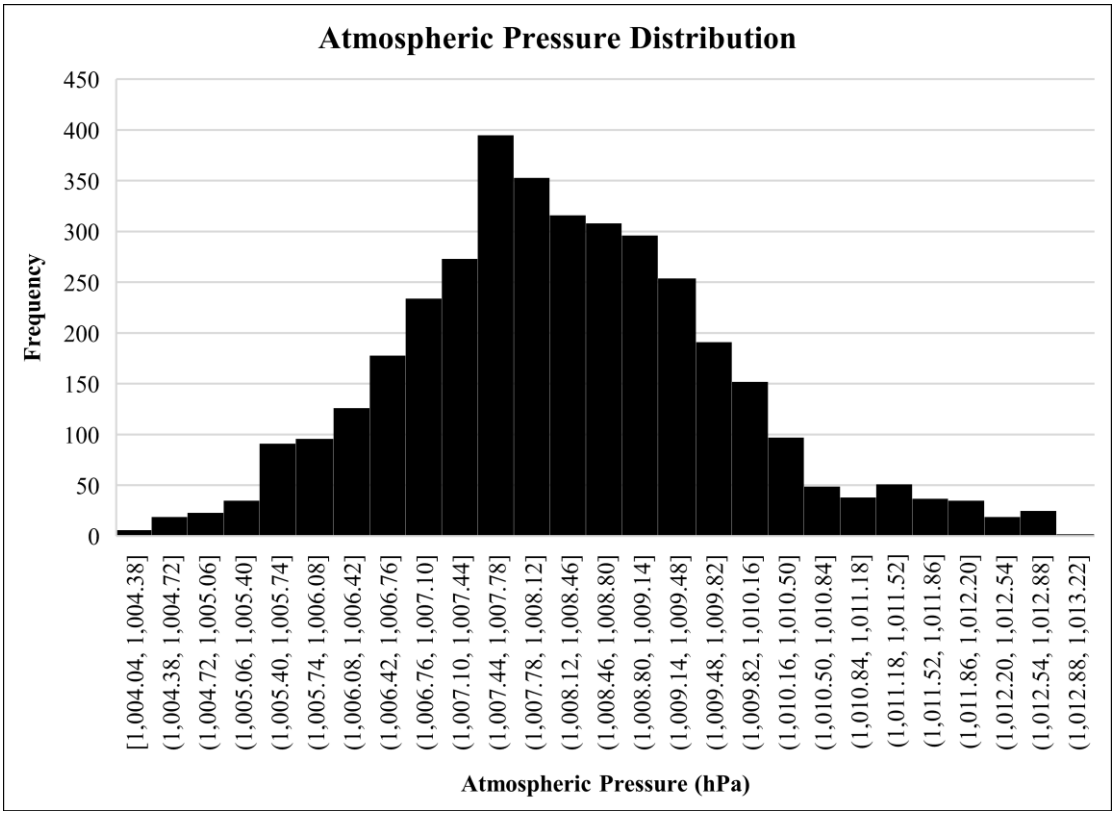


Figure 7.12: Atmospheric Pressure Distribution

Considering the maximum and minimum atmospheric pressure values recorded within the day, Figure 7.13 generated and based on that, it would be able to identify the highest atmospheric pressure recorded on 11th July which is 1013 hPa and the lowest atmospheric pressure was recorded on 31st July which is 1004 hPa.

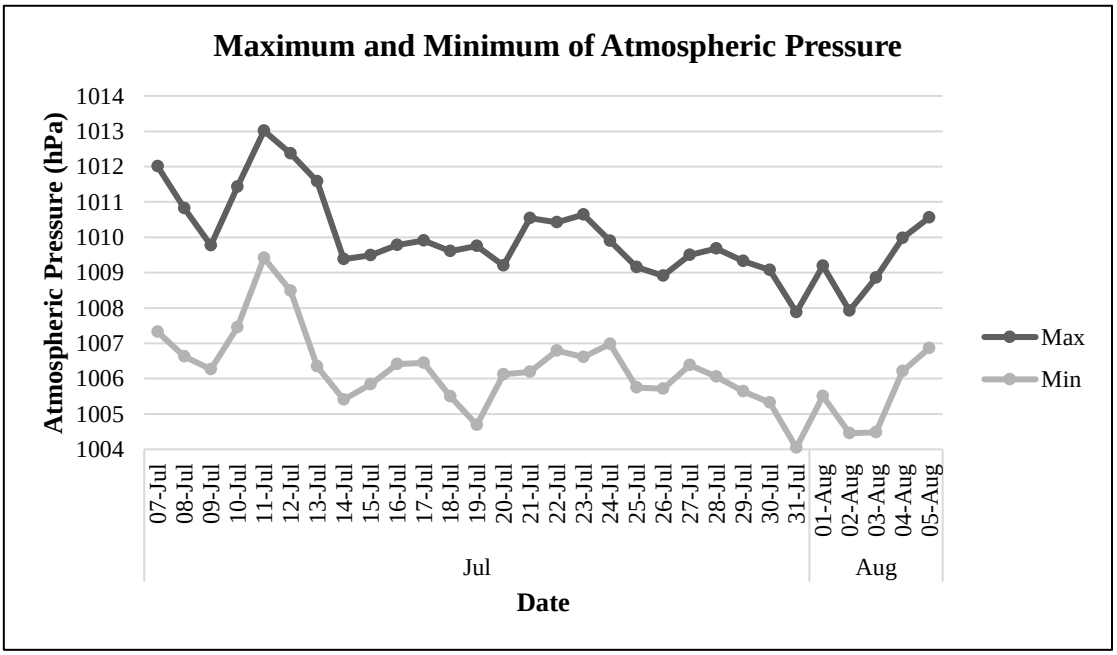


Figure 7.13: Maximum and Minimum Atmospheric Pressure

Considering the highest and the lowest atmospheric pressure recorded time for 30 days, Figures 7.14 and 7.15 was generated. Based on that, it would be able to identify that, the maximum atmospheric pressure will be recorded approximately from 06:15 pm to 03:30 am. The minimum atmospheric pressure will be recorded approximately from 03:00 pm to 07:45 pm.

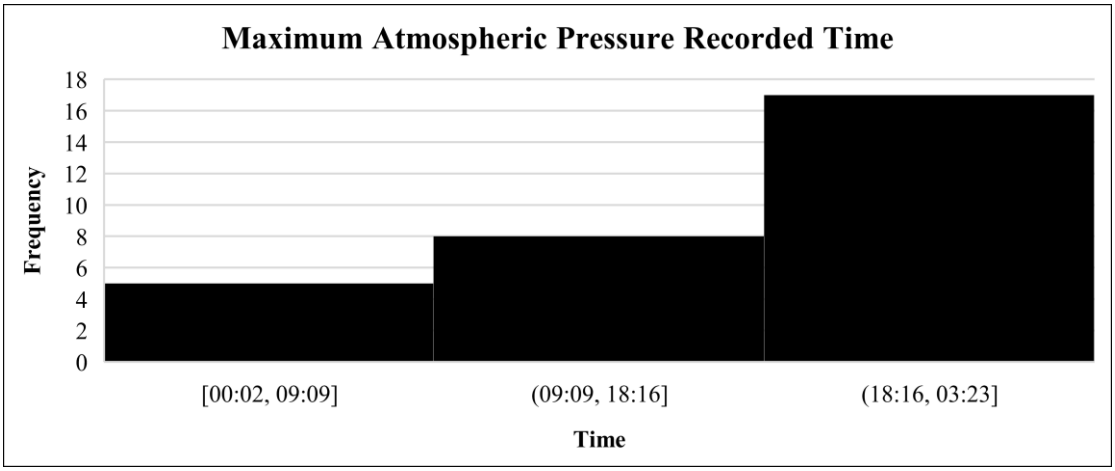


Figure 7.14: Maximum Atmospheric Pressure Recorded Time

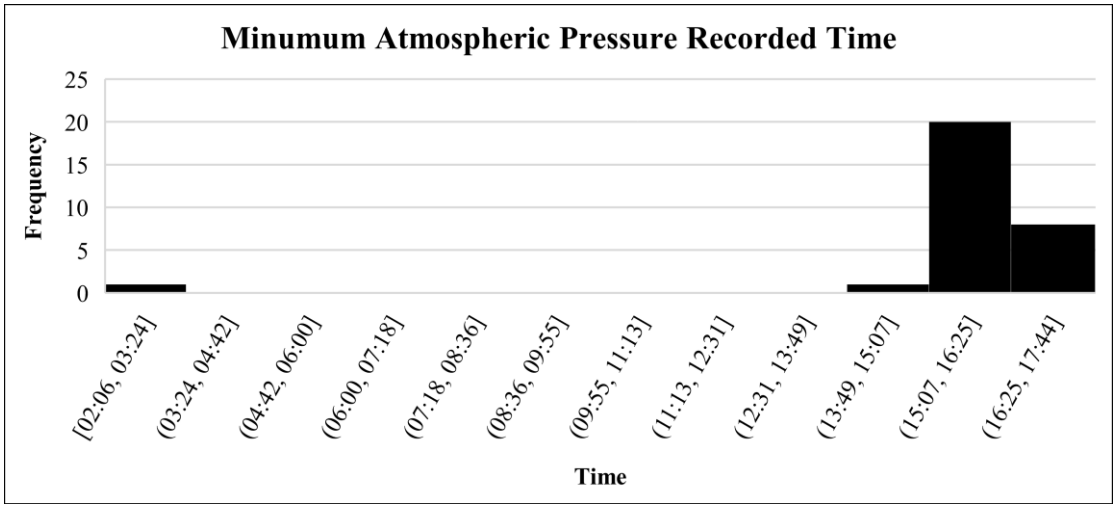


Figure 7.15: Minimum Atmospheric Pressure Recorded Time

7.5 Wind Speed

Using those 30 days of historical data, the average wind speed level for each day was calculated. Figure 7.16 shows how the average wind speed will be changed each day. Based on that it would be able to identify that the average wind speed varied from 2 kilometres per hour (km/h) to 14 km/h. Also, based on the wind speed distribution shown in Figure 7.17, it would be able to identify that from the entire wind speed range,

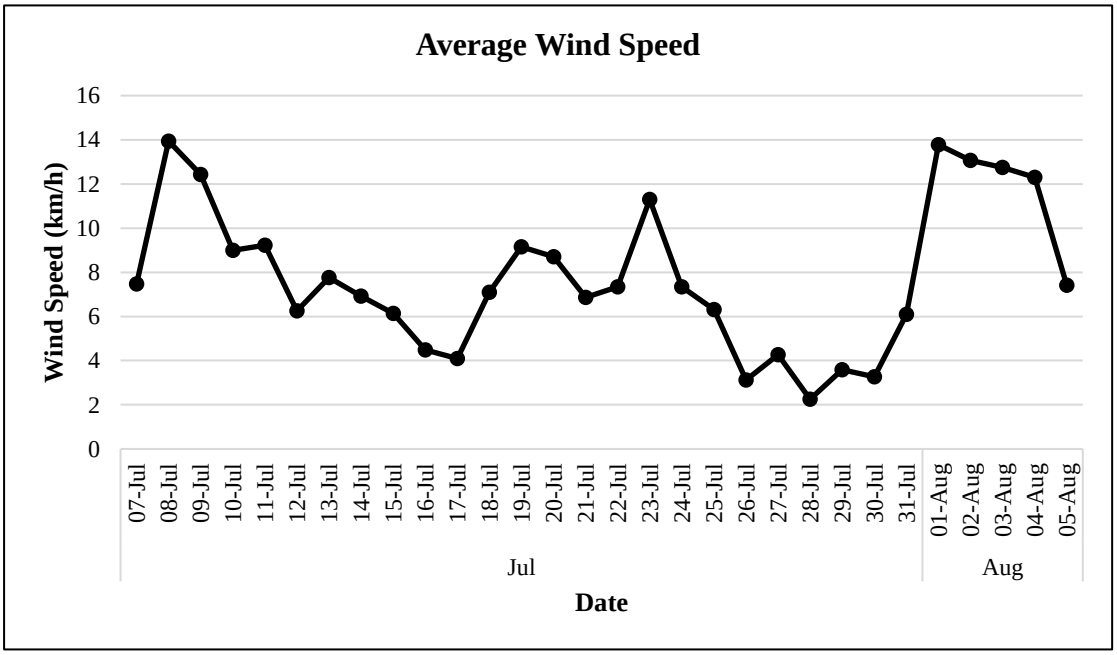


Figure 7.16: Average Wind Speed

most of the recorded wind speed values are from 1.8 km/h to 7.2 km/h with ignoring the 0 km/h wind speed situations.

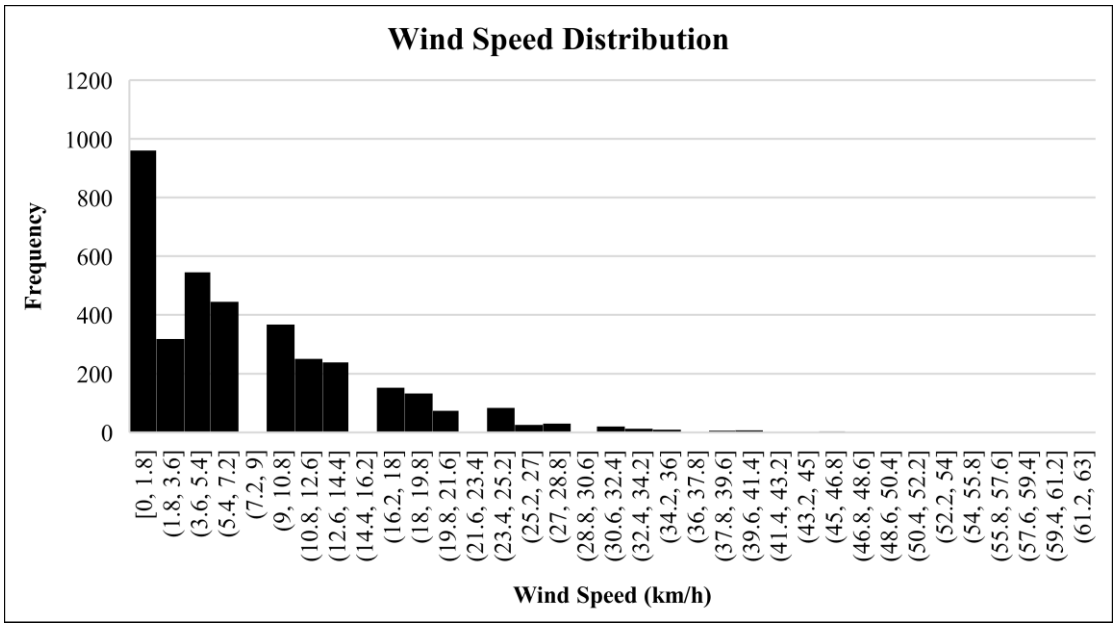


Figure 7.17: Wind Speed Distribution

Considering the maximum and minimum wind speed values recorded within the day, Figure 7.18 was generated and based on that, it would be able to identify the highest wind speed recorded on 04th August which is 62 km/h and the lowest wind speed was scattered through every day and without considering the zero speed, 03rd August recorded 2.4 km/h of wind speed.

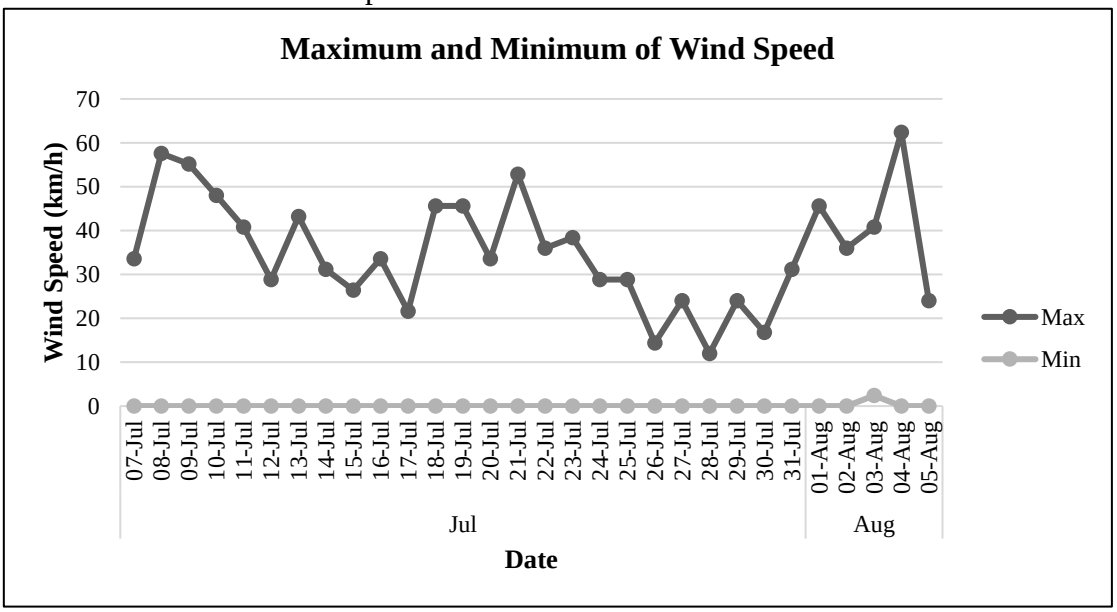


Figure 7.18: Maximum and Minimum Wind Speed

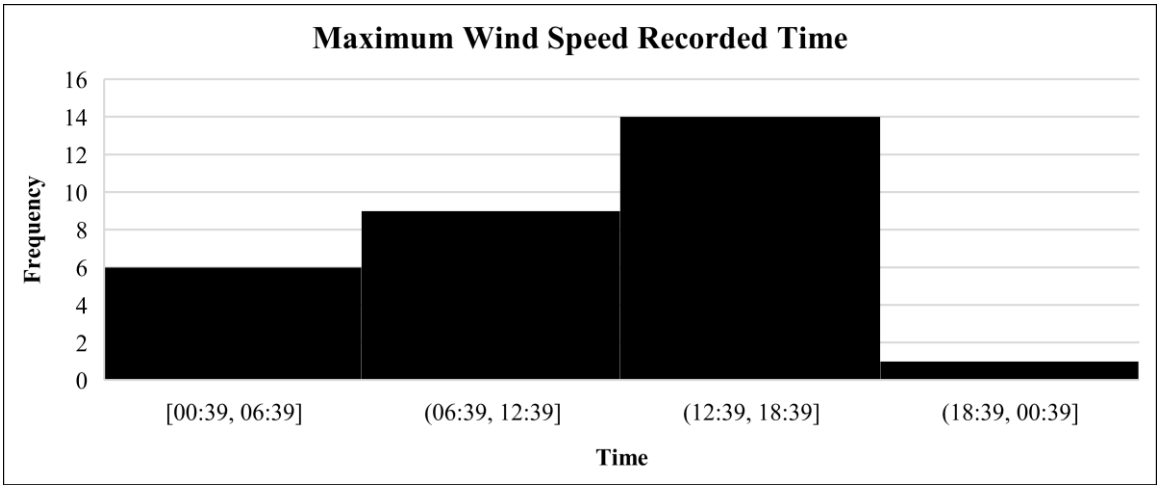


Figure 7.19: Maximum Wind Speed Recorded Time

Considering the highest wind speed recorded time for 30 days, Figure 7.19 was generated. Based on that, it would be able to identify that, the maximum wind speed will be recorded approximately from 12:30 pm to 06:30 pm.

7.6 Wind Direction

Using the wind vane meter, it could be able to identify the wind direction with the resolution of 16 directions. Based on the recorded data, Figure 7.20 would be able to generate for analysing of the most frequent wind direction. Based on that, it could be

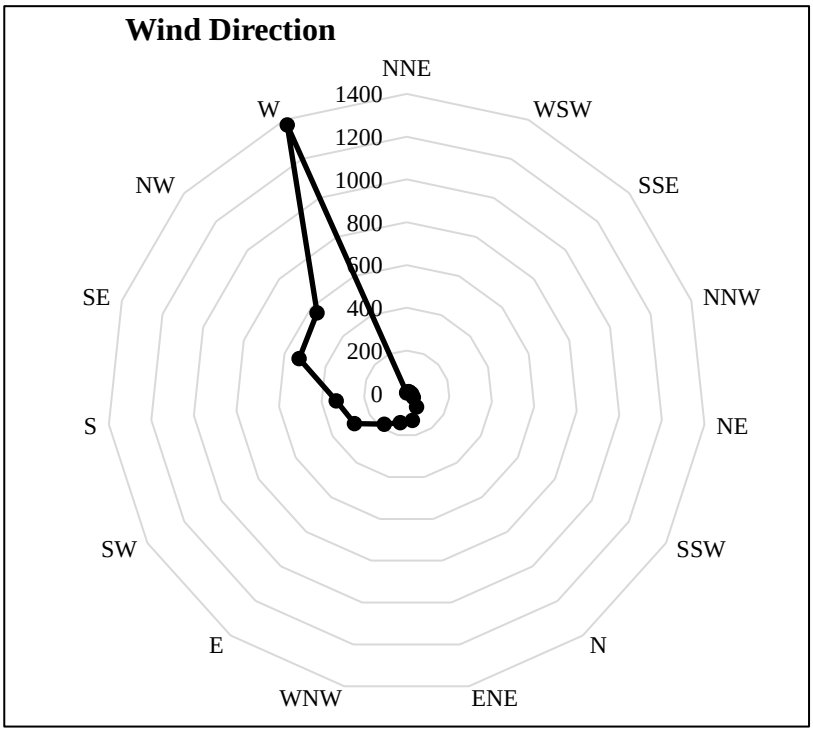


Figure 7.20: Maximum Wind Speed Recorded Time

able to identify that most of the wind is coming from West, North West and South East directions.

7.7 Dew Point

Among the 30 days of historical data, initially, data was used to identify the average dew point for each day. Based on the analysed data, Figure 7.21 would be able to generate. Based on that it would be able to identify that the average dew point varied from 23.5 °C to 25 °C. Also, based on the dew point distribution shown in Figure 7.22, it would be able to identify that from the entire dew point range, most of the recorded dew points are from 24.5 °C to 25.5 °C.

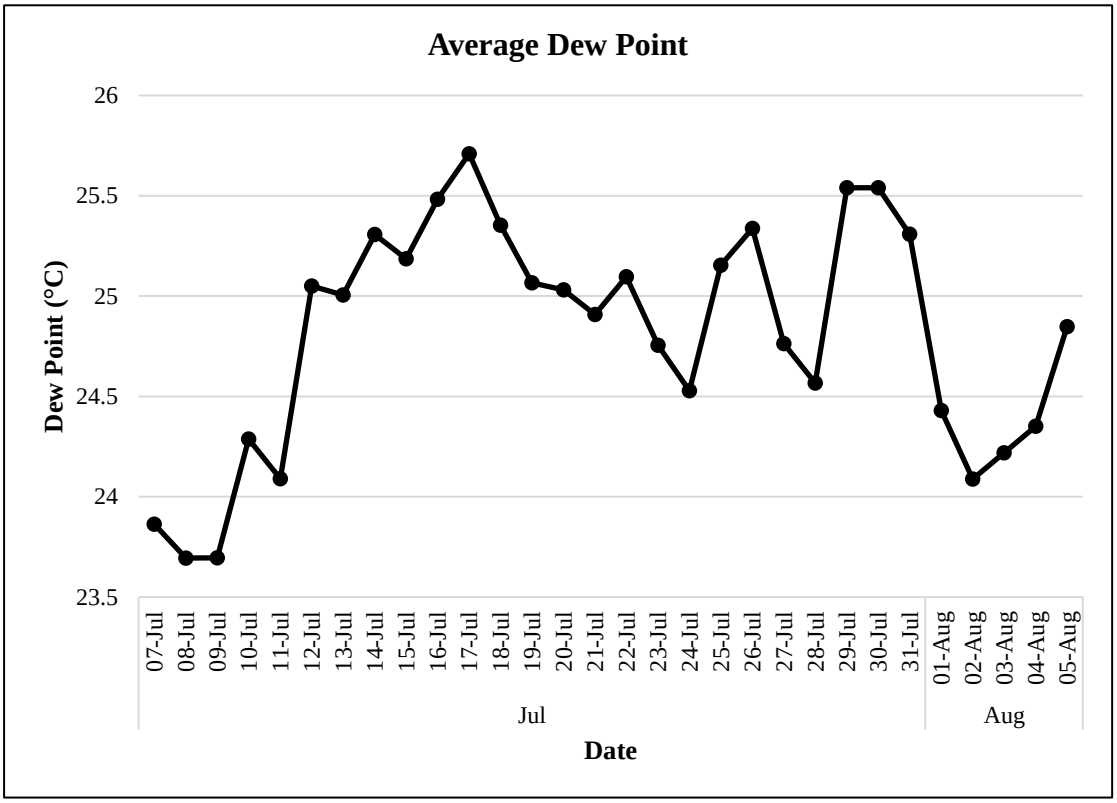


Figure 7.21: Average Dew Point

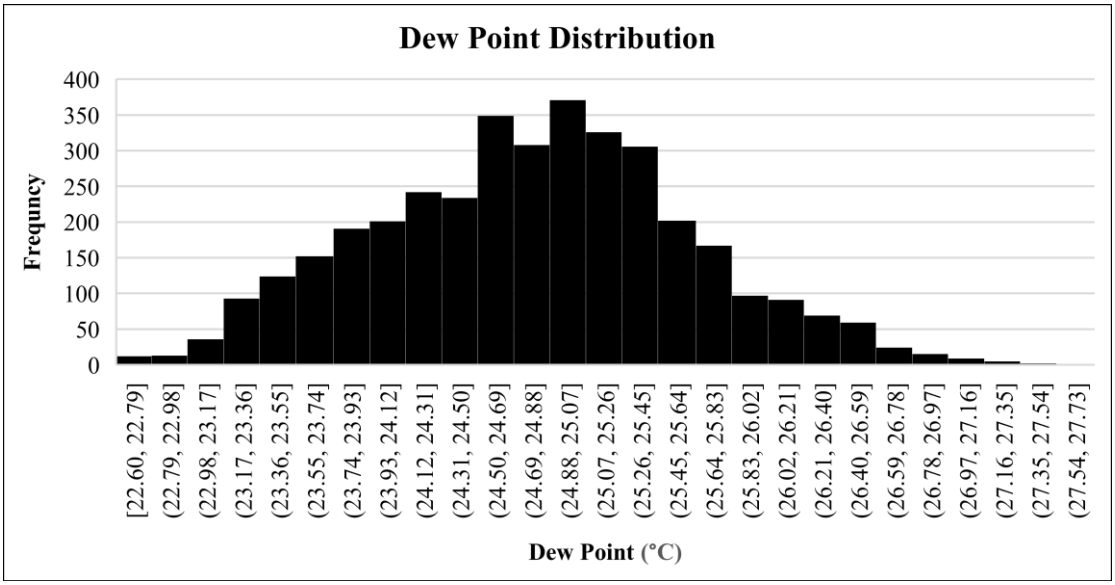


Figure 7.22: Dew Point Distribution

Considering the maximum and minimum dew point values recorded within the day, Figure 7.23 generated and based on that, it would be able to identify the highest dew point recorded on 17th July which is 27.7 °C and the lowest dew point was recorded on 11th July which is 22.6 °C.

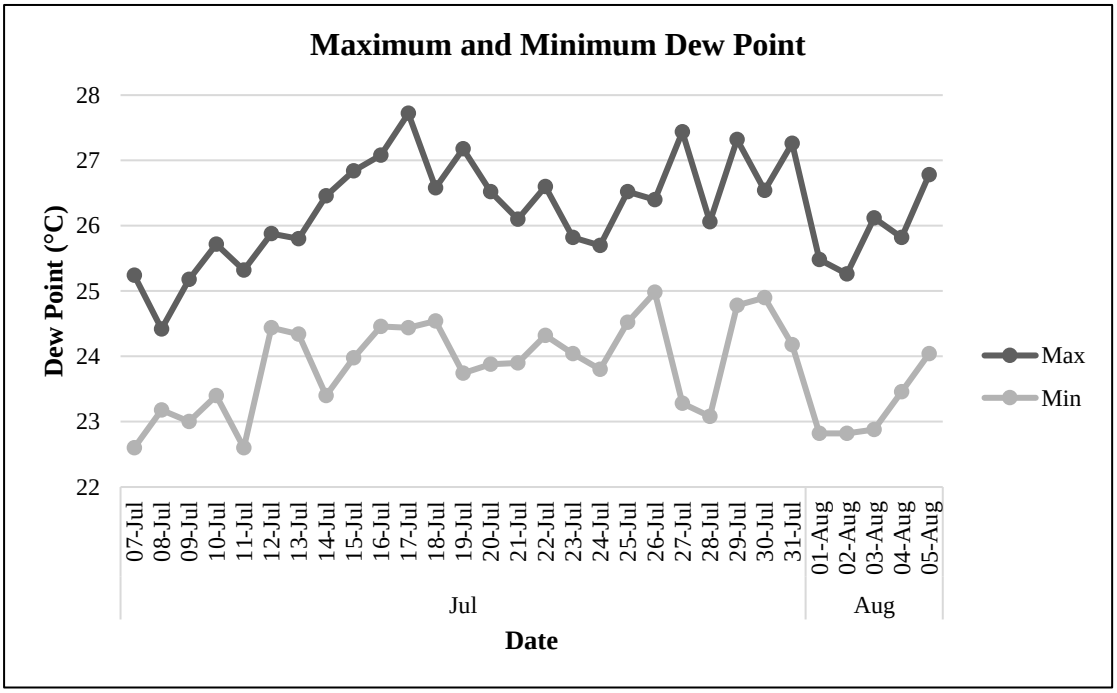


Figure 7.23: Maximum and Minimum Dew Point

Considering the highest and the lowest dew point recorded time for 30 days, Figures 7.24 and 7.25 was generated.

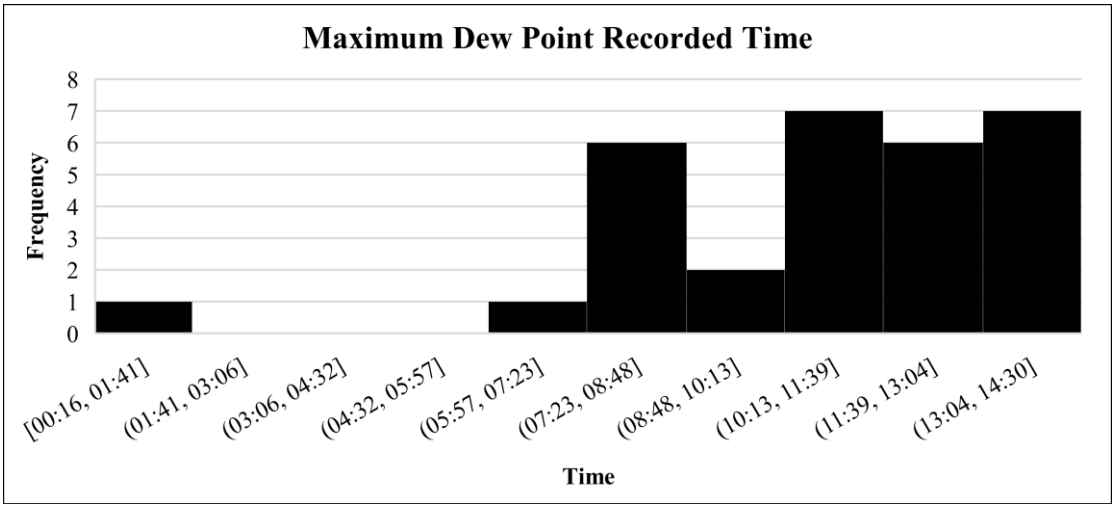


Figure 7.24: Maximum Dew Point Recorded Time

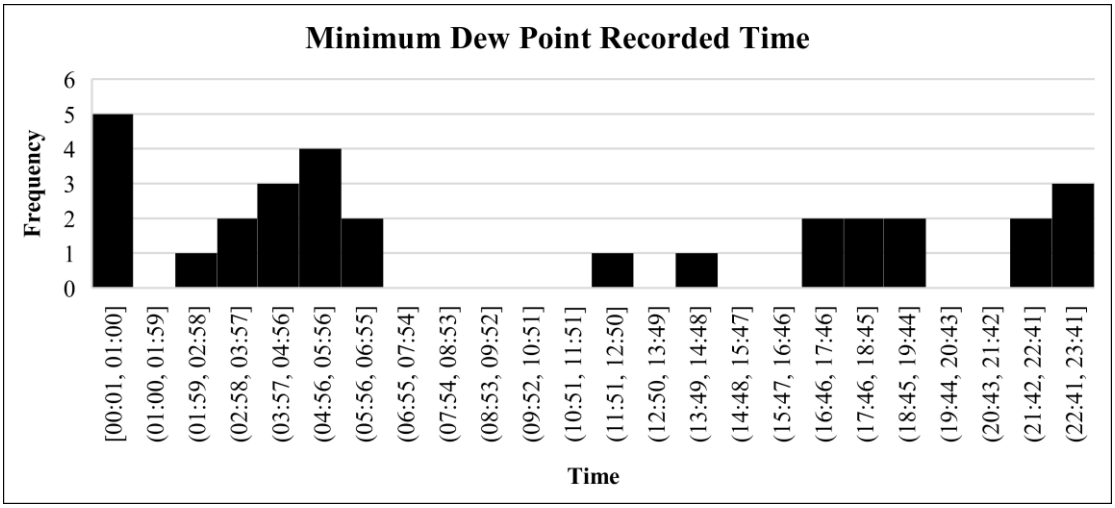


Figure 7.25: Minimum Dew Point Recorded Time

Based on that, it would be able to identify that, the maximum dew point will be recorded approximately from 10:15 am to 02:30 pm. The minimum dew point will be recorded approximately from 09:30 pm to 06:00 am.

7.8 Heat Index

The 30 days of historical data were used to identify the average heat index for each day. Based on the analysed data, Figure 7.26 would be able to generate. Based on that it would be able to identify that the average heat index varied from 27 °C to 33 °C. Also, based on the heat index distribution shown in Figure 7.27, it would be able to identify that from the entire heat index range, most of the recorded heat indexes are from 25 °C to 28.5 °C.

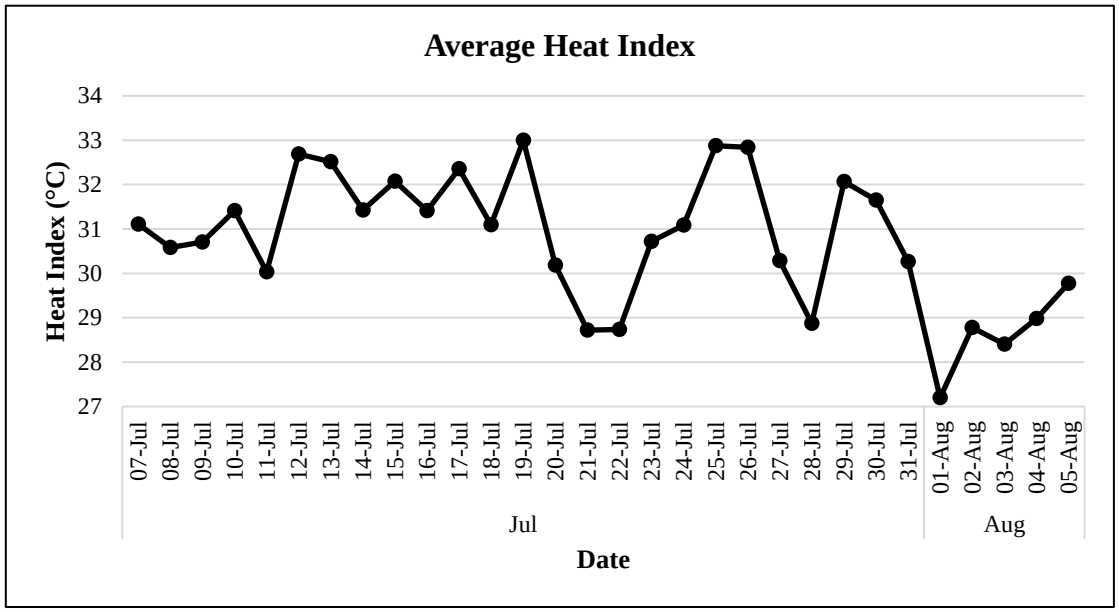


Figure 7.26: Average Heat Index

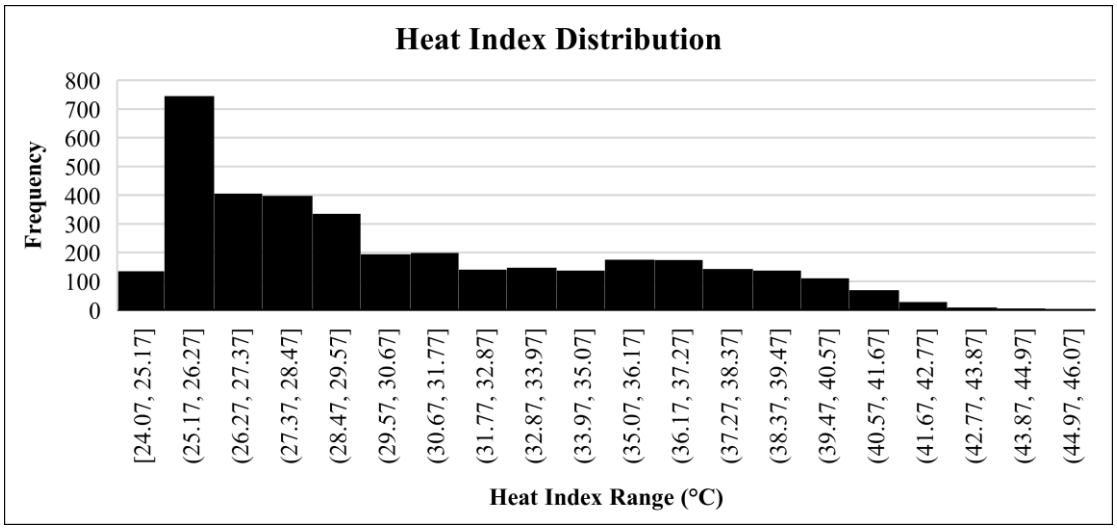


Figure 7.27: Heat Index Distribution

Considering the maximum and minimum heat index values recorded within the day, Figure 7.28 generated and based on that, it would be able to identify the highest heat index recorded on 27th July which is 45.1 °C and the lowest heat index was recorded on 28th July which is 24 °C. based on the maximum heat index recorded, it can be interpreted that heat index is reaching more close to the danger level in the heat index

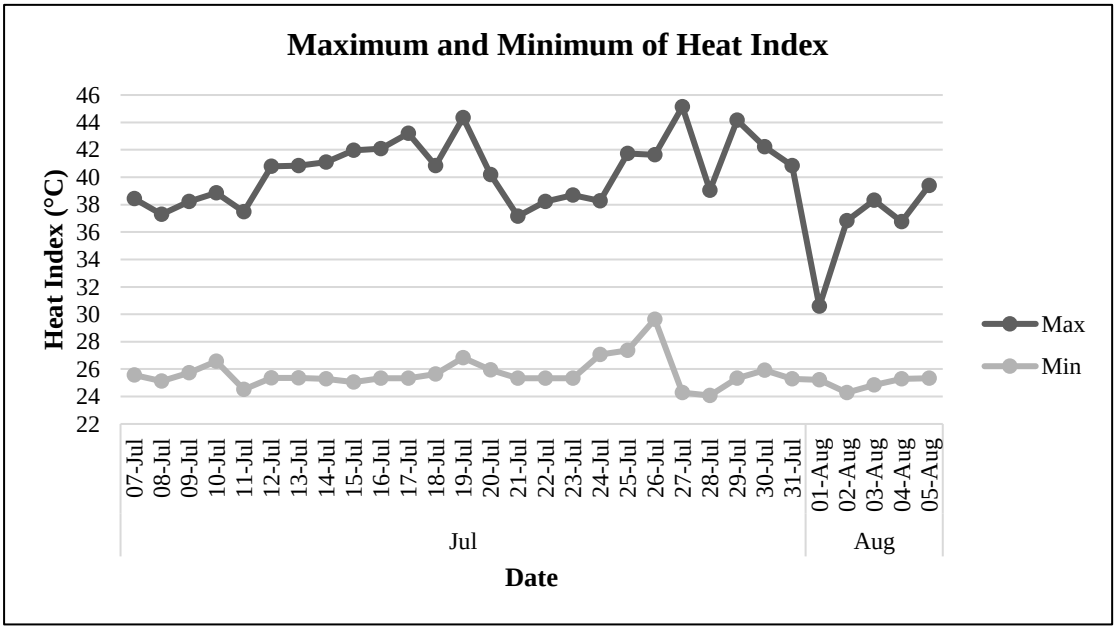


Figure 7.28: Maximum and Minimum Heat Index

warning chart.

Considering the highest and the lowest heat index recorded time for 30 days, Figures 7.29 and 7.30 was generated.

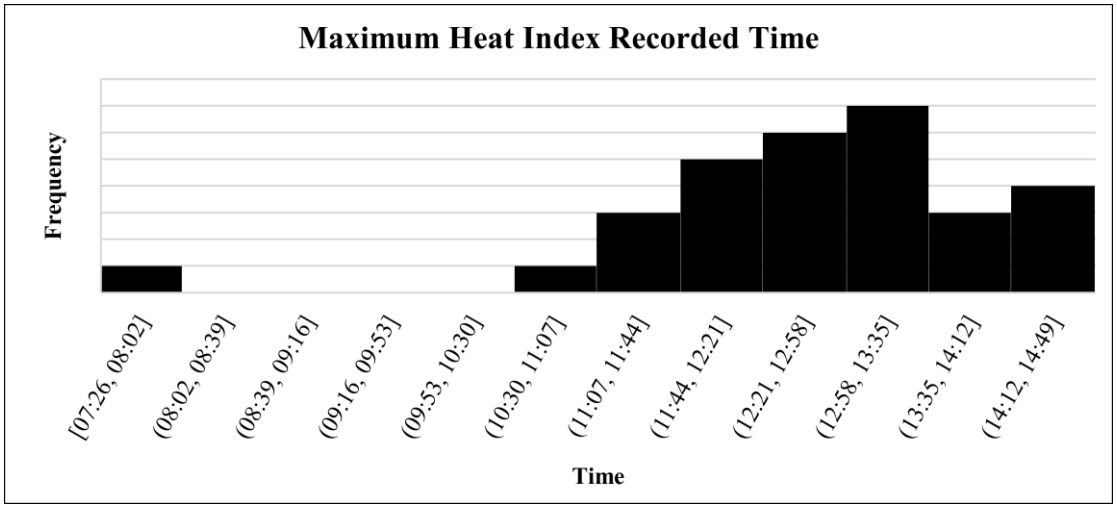


Figure 7.29: Maximum Heat Index Recorded Time

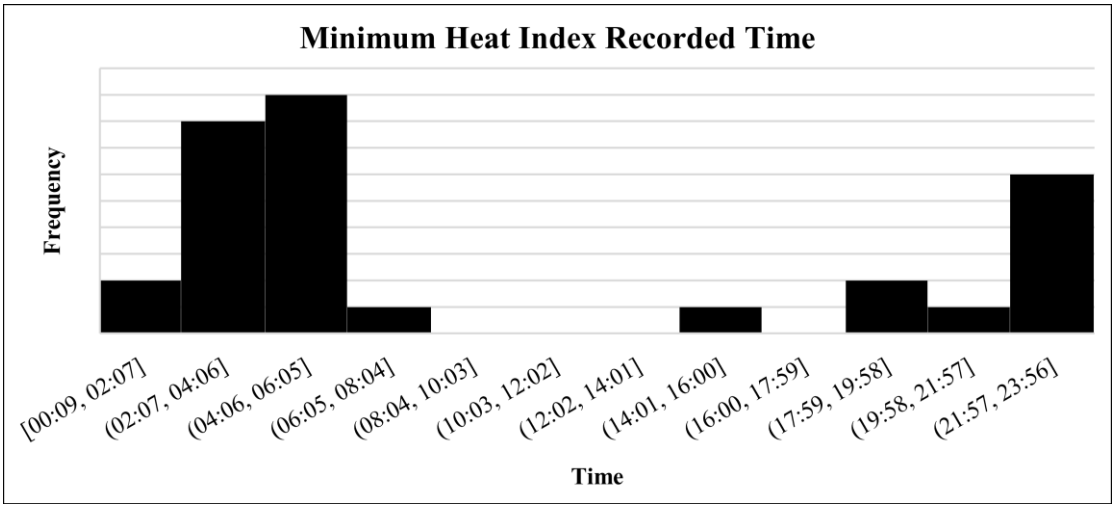


Figure 7.30: Minimum Heat Index Recorded Time

Based on that, it would be able to identify that, the maximum heat index will be recorded approximately from 11:45 am to 01:30 pm. The minimum heat index will be recorded approximately from 02:00 am to 06:00 am.

7.9 Rainfall

Using the rainfall data for 30 days, maximum rainfall for a particular day was identified as the rainfall for the day. Based on that, Figure 7.31 was generated.

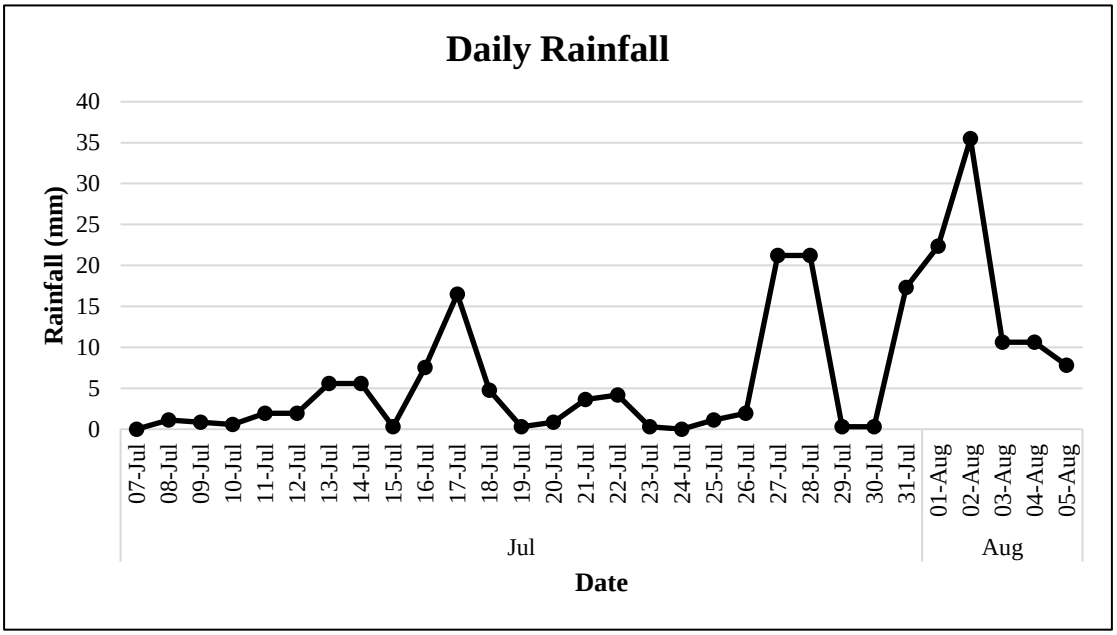


Figure 7.31: Daily Rainfall

Therefore, it would be able to identify that for some days, there is no rainfall and maximum rainfall was recorded on 2nd August which was 35.48 mm. most days, the rainfall was less than 10 mm.

7.10 Weather Prediction

The main purpose of this study is to develop a microcontroller-based weather prediction system. Therefore, the Zambretti algorithm was used to predict the weather. Based on the algorithm, it was capable of giving 26 different weather predictions. Based on the recorded data in the IoT cloud, Figure 7.32 was generated.

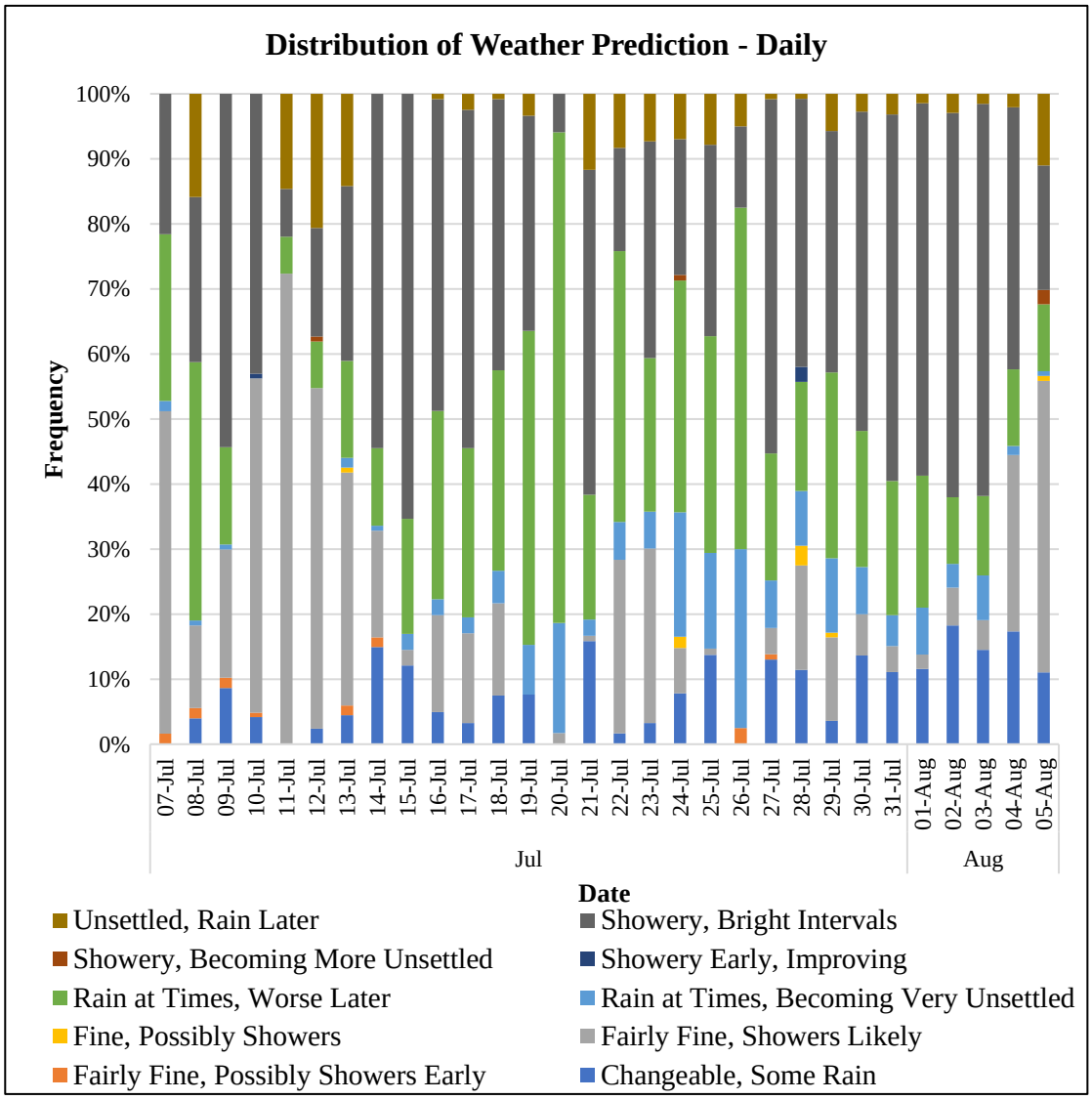


Figure 7.32: Weather Prediction Distribution

Based on that, it would be able to identify that most of the predictions fall under “Showery, Bright Intervals” and “Rain at Times, Worse Later”. With compared to the most frequent item in each day and the rainfall, Table 7.1 would be able to generate. In the accuracy column, “1” represent the rainfall and weather prediction are matched and “0” represent it is not matched.

Table 7.1: Weather Prediction Vs.Rainfall

Date	Prediction	Rainfall (mm)	Accuracy
2022-07-07	Fairly Fine, Showers Likely	0	0
2022-07-08	Rain at Times, Worse Later	1.12	1
2022-07-09	Showery, Bright Intervals	0.84	1
2022-07-10	Fairly Fine, Showers Likely	0.56	1
2022-07-11	Fairly Fine, Showers Likely	1.96	1
2022-07-12	Fairly Fine, Showers Likely	1.96	1
2022-07-13	Fairly Fine, Showers Likely	5.59	1
2022-07-14	Showery, Bright Intervals	5.59	1
2022-07-15	Showery, Bright Intervals	0.28	1
2022-07-16	Showery, Bright Intervals	7.54	1
2022-07-17	Showery, Bright Intervals	16.48	1
2022-07-18	Showery, Bright Intervals	4.75	1
2022-07-19	Rain at Times, Worse Later	0.28	1
2022-07-20	Rain at Times, Worse Later	0.84	1
2022-07-21	Showery, Bright Intervals	3.63	1
2022-07-22	Rain at Times, Worse Later	4.19	1
2022-07-23	Showery, Bright Intervals	0.28	1
2022-07-24	Rain at Times, Worse Later	0	0
2022-07-25	Rain at Times, Worse Later	1.12	1
2022-07-26	Rain at Times, Worse Later	1.96	1
2022-07-27	Showery, Bright Intervals	21.23	1
2022-07-28	Showery, Bright Intervals	21.23	1
2022-07-29	Showery, Bright Intervals	0.28	1
2022-07-30	Showery, Bright Intervals	0.28	1
2022-07-31	Showery, Bright Intervals	17.32	1
2022-08-01	Showery, Bright Intervals	22.35	1
2022-08-02	Showery, Bright Intervals	35.48	1
2022-08-03	Showery, Bright Intervals	10.62	1
2022-08-04	Showery, Bright Intervals	10.62	1
2022-08-05	Fairly Fine, Showers Likely	7.82	1

Based on that, the accuracy of the algorithm was calculated.

Total Occurrences =	30
Correct Occurences =	28
Wrong Occurences =	2
Accuracy of the Algorithm =	93.3%

Therefore, it would be able to identify that the Zambretti algorithm was 93.3% accurate. With compared to other studies done regarding the accuracy of the model, this accuracy level is fall in between others' accuracy levels. Therefore, it can be confirmed that the algorithm was successfully implemented in this study [61] [62].

Chapter 8 - Conclusion And Future Works

8.1 Conclusion

In conclusion, the weather prediction system developed for this study is capable to handle the weather-related sensor to capture the data. Also, it could be able to handle wireless communication without missing the data packets. Considering the solar power-based power supply system, it could be able to power up the outdoor unit for the entire time when the sun is not available and the sunlight is not adequate to generate the required electricity to charge the batteries. Using replacement parts of a commercial weather station gives an additional advantage since there is no requirement for the calibration of sensors and long-lasting durability. Also, using an Arduino Mega 2560 development board helps to manage all the components easily since its massive GPIO pin count. Also, ESP 32 expansion board which was developed as a side project would be a great lifesaver when comes to handling the power and data bus in Arduino Mega 2560 development board and helps to attach all the wires in the indoor unit. Also since it is capable to power the Voltage Common Collector (VCC) and the Ground (GND) lines separately, it protects the ESP 32 development board from drawing higher currents (Amperes) in its voltage regulator and power lines.

Using an ESP 32 development board would be a great choice for the indoor unit due to its faster performance, in-built Wi-Fi and larger memory. Also, nRF24L01 modules were really helpful for wireless communication due to their low power consumption and long range. Using a Nextion display would be the best choice in this entire study. Due to its special capabilities, all the user interfaces become more attractive and easy to use. Also, developing user interfaces become more straightforward. Also, using the Zambretti algorithm was a great success in this study since the predictions are more accurate even though it is developed a long time ago. Finally, it can be concluded that the purpose, objectives and aim were able to achieve at the end of this research study.

8.2 Future Works

This project can be improved by using a GPS module to identify the location, time zone and altitude other than hard code those values. Then it can be improved by using some alert system implemented in the collaboration with the Nextion display since it has audio output too. Also, it can be developed as a mobile phone application that interacts with this weather prediction system. Also, other than using the Zambretti algorithm inside the microcontroller, it can be used in a cloud service platform or it can be used with a different better algorithm that has more accuracy than the algorithm used in this study. Also, it can be implemented with the same user interfaces or better once with a much cheaper and faster display model if possible. Also, instead of ESP 32, it can be used Raspberry Pi Single Board Computer (SBC) and more additional features can be added since it supports Python language and it works like a normal computer.

8.3 Limitations

During this project, few limitations were able to be faced. While programming the ESP 32 development board, due to an issue in a library, it could not be able to use char arrays in the programme. Also, for offline data logging, it could not be able to attach a MicroSD card reader to the ESP 32 development board since the module is using 5V logic levels to operate. There was not any proper solution that was able to be found and there were some recommendations for hardware-level changes in the module. also, it could not be able to attach an RTC module due to all the libraries using char arrays. Using an RTC module caused crashes in the programme during runtime. Also, there are time conversions could not be possible due to using of char arrays for that task.

References

[1] “Weather | Britannica.” https://www.britannica.com/science/weather (accessed Jul. 11, 2021).

[2] “weather | National Geographic Society.” https://www.nationalgeographic.org/encyclopedia/weather/ (accessed Jul. 11, 2021).

[3] “weather forecasting | Methods, Importance, & History | Britannica.” https://www.britannica.com/science/weather-forecasting (accessed Jul. 11, 2021).

[4] “Weather forecasting.” https://www.sciencedaily.com/terms/weather_forecasting.htm (accessed Jul. 11, 2021).

[5] “5 Advantages of weather forecasting.” https://geographypoint.com/2019/04/importance-of-weather-forecasting/ (accessed Jul. 11, 2021).

[6] “How Your Home Weather Station Creates a Weather Forecast.” https://www.instrumentchoice.com.au/news/how-your-home-weather-station-creates-a-weather-forecast (accessed Jun. 15, 2022).

[7] “Interesting Sensors To Add To Your Weather Station Project - Electronics-Lab.com.” https://www.electronics-lab.com/interesting-sensors-to-add-to-your-weather-station-project/ (accessed Jul. 16, 2021).

[8] “10 Best Microcontroller Boards for Engineers and Geeks - Engineering Passion.” https://www.engineeringpassion.com/10-best-microcontroller-boards-for-engineers-and-geeks/ (accessed Jul. 11, 2021).

[9] “Weather Map.” https://www.meteo.gov.lk/index.php?option=com_content&view=article&id=102&Itemid=360&lang=en (accessed Jul. 12, 2021).

[10] K. Krishnamurthi, S. Thapa, L. Kothari, and A. Prakash, “Arduino Based Weather Monitoring System,” *Int. J. Eng. Res. Gen. Sci.*, vol. 3, no. 2.

[11] G. Verma, P. Mittal, and S. Farheen, “Real Time Weather Prediction System Using IOT and Machine Learning,” in *2020 6th International Conference on Signal Processing and Communication (ICSC)*, Mar. 2020, pp. 322–324, doi: 10.1109/ICSC48311.2020.9182766.

[12] “Department of Meteorology - Sri Lanka.” https://www.meteo.gov.lk/index.php?option=com_content&view=article&id=13&Itemid=132&lang=en (accessed Jul. 15, 2021).

[13] “DHT11 basic temperature-humidity sensor + extras : ID 386 : $5.00 : Adafruit Industries, Unique & fun DIY electronics and kits.” https://www.adafruit.com/product/386 (accessed Jul. 15, 2021).

[14] T. Cao-Hoang and C. N. Duy, “Environment monitoring system for agricultural application based on wireless sensor network,” in *7th International Conference on Information Science and Technology, ICIST 2017 - Proceedings*, May 2017, pp. 99–102, doi: 10.1109/ICIST.2017.7926499.

[15] S. A. Nishe, T. A. Tahrin, N. Kamal, S. Hoque, and K. T. Hasan, “Micro-level meteorological data sourcing for accurate weather prediction,” *5th IEEE Reg. 10 Humanit. Technol. Conf. 2017, R10-HTC 2017*, vol. 2018-January, pp. 353–356, Feb. 2018, doi: 10.1109/R10-HTC.2017.8288973.

[16] P. Ashok Baste, S. R. Jadkar, and A. M. Pathak, “Weather Station for Solar PV Power Plant Using Arduino Mega,” in *2021 International Conference on Computer Communication and Informatics (ICCCI)*, Jan. 2021, pp. 1–6, doi: 10.1109/ICCCI50826.2021.9402478.

[17] “BMP180 Digital pressure sensor.” https://cdn-shop.adafruit.com/datasheets/BST-BMP180-DS000-09.pdf (accessed Jul. 16, 2021).

[18] B. Sensortec, “BME280 Combined humidity and pressure sensor.” https://cdn-

shop.adafruit.com/datasheets/BST-BME280_DS001-10.pdf (accessed Jul. 16, 2021).

[19] M. Farhat, M. Abdul-Niby, M. Abdullah, and A. Nazzal, “A Low Cost Automated Weather Station for Real Time Local Measurements,” *Eng. Technol. Appl. Sci. Res.*, vol. 7, no. 3, pp. 1615–1618, Jun. 2017, doi: 10.48084/ETASR.1187.

[20] “SparkFun Weather Shield - DEV-13956 - SparkFun Electronics.” https://www.sparkfun.com/products/13956 (accessed Jan. 14, 2022).

[21] “Xtrinsic MPL3115A2 I2C Precision Altimeter,” *Freescale Semiconductor*, 2013. https://cdn-shop.adafruit.com/datasheets/1893_datasheet.pdf (accessed Jan. 14, 2022).

[22] A. Y. Ardiansyah, R. Sarno, and O. Giandi, “Rain detection system for estimate weather level using Mamdani fuzzy inference system,” in *2018 International Conference on Information and Communications Technology (ICOIACT)*, Mar. 2018, vol. 2018-Janua, pp. 848–854, doi: 10.1109/ICOIACT.2018.8350711.

[23] “Analysis of local weather radar data in support of sewer system modelling.” https://www.researchgate.net/publication/318108618_Analysis_of_local_weather_radar_data_in_support_of_sewer_system_modelling (accessed Jun. 15, 2022).

[24] V. Abhyankar, A. G. Singh, P. Paul, A. Mehta, and S. Vidhya, “Portable Autonomous Rain Prediction Model Using Machine Learning Algorithm,” in *2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN)*, Mar. 2019, pp. 1–4, doi: 10.1109/ViTECoN.2019.8899704.

[25] H. Saini, A. Thakur, S. Ahuja, N. Sabharwal, and N. Kumar, “Arduino based automatic wireless weather station with remote graphical application and alerts,” in *2016 3rd International Conference on Signal Processing and Integrated Networks (SPIN)*, Feb. 2016, pp. 605–609, doi:

10.1109/SPIN.2016.7566768.

[26] B. Idjeri, M. Laghrouche, and J. Boussey, “Wind Measurement Based on MEMS Micro-Anemometer With High Accuracy Using ANN Technique,” *IEEE Sens. J.*, vol. 17, no. 13, pp. 4181–4188, Jul. 2017, doi: 10.1109/JSEN.2017.2701502.

[27] A. Felipe *et al.*, “DESIGN AND IMPLEMENTATION OF A METEOROLOGICAL STATION WITH A WIRELESS DATA ACQUISITION SYSTEM (WDAS),” vol. 16, no. 5, 2021.

[28] D. Satria, S. Yana, R. Munadi, and S. Syahreza, “Prototype of Google Maps-Based Flood Monitoring System Using Arduino and GSM Module,” *Int. Res. J. Eng. Technol.*, 2017.

[29] M. S. Monteiro, F. L. de Caldas Filho, L. A. Barbosa, L. M. C. E. Martins, J. T. M. de Menezes, and D. A. da Silva Filho, “University Campus Microclimate Monitoring Using IoT,” in *2019 Workshop on Communication Networks and Power Systems (WCNPS)*, Oct. 2019, pp. 1–5, doi: 10.1109/WCNPS.2019.8896242.

[30] Z. K. Hussein, H. J. Hadi, M. R. Abdul-Mutaleb, and Y. S. Mezaal, “Low cost smart weather station using Arduino and ZigBee,” *TELKOMNIKA (Telecommunication Comput. Electron. Control.*, vol. 18, no. 1, p. 282, Feb. 2020, doi: 10.12928/telkomnika.v18i1.12784.

[31] E. Murdyantoro, R. Setiawan, I. Rosyadi, A. W. Nugraha, H. Susilawati, and Y. Ramadhani, “Prototype weather station uses LoRa wireless connectivity infrastructure,” *J. Phys. Conf. Ser.*, vol. 1367, p. 012089, Nov. 2019, doi: 10.1088/1742-6596/1367/1/012089.

[32] R. Sidqi, R. Rynaldo, S. H. Suroso, and R. Firmansyah, “Arduino Based Weather Monitoring Telemetry System Using NRF24L01+,” doi: 10.1088/1757-899X/336/1/012024.

[33] T. Vijaya Kumar, L. Yasaswini, C. H. Lavanya, and K. Suraj Kalyan,

"RENEWABLE POWERED PORTABLE WEATHER MONITORING SYSTEM," *Int. Res. J. Eng. Technol.*, 2020.

[34] A. Salman Abd-Elmajed and A. Eldeen Abd-Allah Awouda, "A Microcontrooller-Based Weather Prediction System using the Sliding Window Algorithm," 2020.

[35] "Home - Nextion." https://nextion.tech/ (accessed Jul. 13, 2021).

[36] "DOWNLOAD - Nextion." https://nextion.tech/nextion-editor/#_section1 (accessed Jul. 16, 2021).

[37] "Arduino vs ESP8266 vs ESP32 Microcontroller Comparison." https://diyi0t.com/technical-datasheet-microcontroller-comparison/ (accessed Jul. 16, 2021).

[38] L. Varghese, G. Deepak, and A. Santhanavijayan, "An IoT Analytics Approach for Weather Forecasting using Raspberry Pi 3 Model B+," *2019 15th Int. Conf. Inf. Process. Internet Things, ICINPRO 2019 - Proc.*, Dec. 2019, doi: 10.1109/ICINPRO47689.2019.9092107.

[39] M. Kusriyanto and A. A. Putra, "Weather Station Design Using IoT Platform Based On Arduino Mega," in *2018 International Symposium on Electronics and Smart Devices (ISESD)*, Oct. 2018, pp. 1–4, doi: 10.1109/ISESD.2018.8605456.

[40] J. B. Young, "AN3914, Modern Altimeter and Barometer System using the MPL115A."

[41] Y. Radhika and M. Shashi, "Atmospheric Temperature Prediction using Support Vector Machines," *Int. J. Comput. Theory Eng.*, pp. 55–58, 2009, doi: 10.7763/IJCTE.2009.V1.9.

[42] T. P. Fowdur, Y. Beeharry, V. Hurbungs, V. Bassoo, V. Ramnarain-Seetohul, and E. C. M. Lun, "Performance analysis and implementation of an adaptive real-time weather forecasting system," *Internet of Things*, vol. 3–4, pp. 12–33,

Oct. 2018, doi: 10.1016/j.iot.2018.09.002.

[43] Gopinath N, Vinodh S, Prashanth P, Jayasuriya A, and Deasione S, “Weather Prediction using Machine Learning and IOT,” *Int. J. Eng. Adv. Technol.*, vol. 9, no. 4, pp. 2094–2098, Apr. 2020, doi: 10.35940/ijeat.D9130.049420.

[44] R. S. Vignesh, A. Sivakumar, M. Shyam, and J. Yogapriya, “Deep Reinforcement Learning Based Weather Monitoring Systemusing Arduino for Smart Environment,” *Int. J. Recent Technol. Eng.*, no. 4, pp. 2277–3878, 2019, doi: 10.35940/ijrte.D8261.118419.

[45] E. S. Semenov, G. S. Ivanchenko, A. V Kharchenko, and R. V Kolobanov, “Mobile weather station based on ATmega2560 microprocessor,” *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 537, no. 3, p. 032086, May 2019, doi: 10.1088/1757-899X/537/3/032086.

[46] “Arduino - AboutUs.” https://www.arduino.cc/en/Main/AboutUs (accessed Sep. 19, 2021).

[47] V. Georgieva and Y. Shunin, “‘Arduino’ – open-source physical computing Input / Output board,” *14th Int. Sci. Conf. Inf. Technol. Manag. 2016*, 2016.

[48] Tukhtanazarovich Jumabayev Abdulkhamid, “ADVANTAGE OF THE ARDUINO PLATFORM IN FORMING CREATIVE SKILLS IN YOUTH,” *JournalNX- A Multidiscip. Peer Rev. J.*, vol. 7, no. 7, Jul. 2021.

[49] “Arduino - Products.” https://www.arduino.cc/en/Main/Products (accessed Sep. 19, 2021).

[50] “Arduino Leonardo with Headers - DEV-11286 - SparkFun Electronics.” https://www.sparkfun.com/products/11286 (accessed Sep. 19, 2021).

[51] “Arduino Pro.” https://www.arduino.cc/pro (accessed Sep. 20, 2021).

[52] “ESP32 Series Datasheet,” 2022. https://www.espressif.com/en/support/download/documents. (accessed Jun. 18, 2022).

[53] “Arduino Mega 2560 Rev3 — Arduino Online Shop.” https://store-usa.arduino.cc/collections/boards/products/arduino-mega-2560-rev3 (accessed Jun. 18, 2022).

[54] “DPS310 - Digital XENSIV TM Barometric Pressure Sensor for Portable Devices Product Description.” .

[55] N. Semiconductors, “MPL3115A2, I2C precision pressure sensor with altimetry.” .

[56] “SNx4HC14 Hex Inverters with Schmitt-Trigger Inputs,” 2021. www.ti.com (accessed Dec. 18, 2021).

[57] “Weather_Meter.” https://cdn.sparkfun.com/assets/d/1/e/0/6/DS-15901-Weather_Meter.pdf (accessed Jul. 16, 2021).

[58] “DS3231.” https://datasheets.maximintegrated.com/en/ds/DS3231.pdf (accessed Jun. 19, 2022).

[59] “Dew Point vs Humidity.” https://www.weather.gov/arx/why_dewpoint_vs_humidity (accessed Jun. 20, 2022).

[60] “What is the heat index?” https://www.weather.gov/ama/heatindex (accessed Jun. 20, 2022).

[61] “Zambretti Algorithm for Weather Forecasting - SAS Support Communities.” https://communities.sas.com/t5/SAS-Analytics-for-IoT/Zambretti-Algorithm-for-Weather-Forecasting/td-p/679487 (accessed Jul. 13, 2021).

[62] “DrKFS.net: A integer forecasting algorithm for barometers.” https://integritext.net/DrKFS/zambretti.htm (accessed Jul. 13, 2021).

Appendix

10.1 Arduino Code for Outdoor Unit

```
#include "DHT.h"
#include <Wire.h>
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <Adafruit_MPL3115A2.h>
#include "RTClib.h"


#define rain_Pin_Intrpt 3 // interrupt pin tied to rain gauge
#define rainfall_Per_Tip 0.2794
#define wind_Direction_Pin A0
#define debounce_Time 15
#define interval 1000
#define anemometer_Pin_Intrpt 2
#define DHTPIN 5
#define DHTTYPE DHT21

RTC_DS3231 rtc;

volatile int tip_Count = 0;
float rainfall = 0;

float degree_of_Direction[] = {112.5, 67.5, 90, 157.5, 135, 202.5,
180, 22.5, 45, 247.5, 225, 337.5, 0, 292.5, 315, 270};
char* direction_Name[] = {"ESE", "ENE", "E", "SSE", "SE", "SSW", "S",
"NNE", "NE", "WSW", "SW", "NNW", "N", "WNW", "NW", "W"};
float Weather_Data_01[8];
int minimum_Read[] = {63, 80, 89, 120, 175, 232, 273, 385, 438, 569,
613, 667, 746, 812, 869, 931};
int maximum_Read[] = {69, 88, 98, 133, 194, 257, 301, 426, 484, 612,
661, 737, 811, 868, 930, 993};
int current_Read = 0;
float wind_Angle = 0;
char* direction_in_Text;
int directionID, i;


unsigned long next_Calc;
unsigned long timer;
volatile int anemometer_RPM;
volatile unsigned long last_Micros_Animomtr;
float humidity, temperature, heat_Index, wind_Speed;
float pressure;
int last_Date;

DHT dht(DHTPIN, DHTTYPE);
RF24 radio(7, 8); // CE, CSN
const byte address[6] = "00001";
Adafruit_MPL3115A2 baro;

void setup() {
```

```
  Serial.begin(9600);
  dht.begin();
  baro.begin();
  radio.begin();
  radio.openWritingPipe(address);
  radio.setPALevel(RF24_PA_MIN);
  radio.stopListening();

  pinMode(10, OUTPUT);
  digitalWrite(10, HIGH);
  rtc.begin();
  DateTime now = rtc.now();
  last_Date = now.day();

  pinMode(rain_Pin_Intrpt, INPUT);
  attachInterrupt(digitalPinToInterrupt(rain_Pin_Intrpt),
rainfall_Count, RISING);

  pinMode(anemometer_Pin_Intrpt, INPUT);
  attachInterrupt(digitalPinToInterrupt(anemometer_Pin_Intrpt),
countAnemometer, FALLING);

}

void loop() {

  DateTime now = rtc.now();

  current_Read = analogRead(wind_Direction_Pin);
  for (i = 0; i <= 15; i++) {
    if (current_Read >= minimum_Read[i] && current_Read <=
maximum_Read[i]) {
      direction_in_Text = direction_Name[i];
      wind_Angle = degree_of_Direction[i];
      directionID = i;
      break;
    }
  }

  timer = millis();

  if (timer > next_Calc) {
    next_Calc = timer + interval;
    //UPDATE ALL VALUES
    wind_Speed = read_Wind_Speed();
    //    Serial.print("\t Wind Speed: " + String(wind_Speed) +
"\t");
  }

  humidity = dht.readHumidity();
  temperature = dht.readTemperature();
  heat_Index = dht.computeHeatIndex(temperature, humidity, false);
  pressure = baro.getPressure();
```

```
  Weather_Data_01[0] = rainfall;
  Weather_Data_01[1] = directionID;
  Weather_Data_01[2] = wind_Angle;
  Weather_Data_01[3] = wind_Speed;
  Weather_Data_01[4] = temperature;
  Weather_Data_01[5] = humidity;
  Weather_Data_01[6] = heat_Index;
  Weather_Data_01[7] = pressure;

  for (int j = 0; j <= 7; j++) {
    Serial.print(Weather_Data_01[j]);
    Serial.print("\t");
  }

  Serial.println();
  radio.write(&Weather_Data_01, sizeof(Weather_Data_01));


  if (now.day() != last_Date) {
    tip_Count = 0;
    rainfall = 0;
    last_Date = now.day();
  }

  delay(1000);
}

void rainfall_Count() {
  tip_Count = tip_Count + 1;
  rainfall = tip_Count * rainfall_Per_Tip;
}

//returns the wind speed since the last calcInterval.
float read_Wind_Speed() {
  //  unsigned char i;
  float speed_Of_Wind = 24000.0; // one turn = 2.4km per hour; modify
as necessary for your instrument
  speed_Of_Wind = speed_Of_Wind * anemometer_RPM;
  speed_Of_Wind = speed_Of_Wind / 10000.0;
  anemometer_RPM = 0;
  return (float)speed_Of_Wind;
}

void countAnemometer() {
  if ((long)(micros() - last_Micros_Animomtr) >= debounce_Time *
1000) {
    anemometer_RPM = anemometer_RPM + 1;
    last_Micros_Animomtr = micros();
  }

}
```

10.2 Arduino Code for Indoor Unit

```
#include "EasyNextionLibrary.h"

EasyNex myNex(Serial);
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <Wire.h>
#include <WiFi.h>
#include "time.h"
#include <Dusk2Dawn.h>
#include "AdafruitIO_WiFi.h"

#define WIFI_SSID "Dialog 4G"
#define WIFI_PASS "ka1024v6"

#define IO_USERNAME  "akilakavinda"
#define IO_KEY       "aio_wNYe59GwfTd4T3IagtT6viykJeUp"

AdafruitIO_WiFi io(IO_USERNAME, IO_KEY, WIFI_SSID, WIFI_PASS);

AdafruitIO_Feed *temperature_val = io.feed("temperature_val");
AdafruitIO_Feed *humidity_val = io.feed("humidity_val");
AdafruitIO_Feed *pressure_val = io.feed("pressure_val");
AdafruitIO_Feed *wind_direction_val = io.feed("wind_direction_val");
AdafruitIO_Feed *wind_speed_val = io.feed("wind_speed_val");
AdafruitIO_Feed *precipitation_val = io.feed("precipitation_val");
AdafruitIO_Feed *dew_point_val = io.feed("dew_point_val");
AdafruitIO_Feed *heat_index_val = io.feed("heat_index_val");
AdafruitIO_Feed *weather_prediction_val =
io.feed("weather_prediction_val");


const char* ntpServer = "pool.ntp.org";
const long  gmtOffset_sec = 19800;
const int   daylightOffset_sec = 0;
Dusk2Dawn Home(6.8052256, 79.9625526, +5.5);
const float altitude_meters = 24.0;
float pressureArray[18] = {0};
float current_Pressure = 0;
float previous_Pressure = 0;
unsigned long timer, timer_Save_Data, timer_Update_Cloud;
byte counter = 0;
int mp;

char DD[3];
char MM[3];
char MMM[15];
char YY[5];
char HH[3];
char Mn[3];
char half[3];
char sec[3];
char HH24[3];
```

```
float Weather_Data_01[8];
float rainfall, wind_Angle, wind_Speed, temperature, humidity,
heat_Index, pressure, dew_Point, sea_Level_Pressure, zambretti_Calc;
int directionID, D, M, Y;
int last_Date;
char* direction_Name[] = {"ESE", "ENE", "E", "SSE", "SE", "SSW", "S",
"NNE", "NE", "WSW", "SW", "NNW", "N", "WNW", "NW", "W"};

String wind_Direction, sunriseTxt, sunsetTxt, solarnoonTxt,
sunriseTxtF, sunsetTxtF, solarnoonTxtF, moonPhase, dateText,
timeText, YYTxt, MMTxt, DDTxt, HHTxt, MnTxt, SSTxt, halfTxt;

RF24 radio(4, 5); // CE, CSN

const byte address[6] = "00001";

int weather_Prediction, timeHour;
String weather[33] = {"Error Values", "Settled Fine", "Fine Weather",
"Fine, Becoming Less Settled", "Fairly Fine, Showery Later",
"Showery, Becoming More Unsettled", "Unsettled, Rain Later", "Rain at
Times, Worse Later", "Rain at Times, Becoming Very Unsettled", "Very
Unsettled, Rain", "Settled Fine", "Fine Weather", "Fine, Possibly
Showers", "Fairly Fine, Showers Likely", "Showery, Bright Intervals",
"Changeable, Some Rain", "Unsettled, Rain at Times", "Rain at
Frequent Intervals", "Very Unsettled, Rain", "Stormy, Much Rain",
"Settled Fine", "Fine Weather", "Becoming Fine", "Fairly Fine,
Improving", "Fairly Fine, Possibly Showers Early", "Showery Early,
Improving", "Changeable, Mending", "Rather Unsettled, Clearing
Later", "Unsettled, Probably Improving", "Unsettled, Short Fine
Intervals", "Very Unsettled, Finer at Times", "Stormy, Possibly
Improving", "Stormy, Much Rain"};
String trend = "Steady";
String weather_Prediction_Status;

float temperature_Min, temperature_Avg, temperature_Max,
temperature_Total, humidity_Min, humidity_Avg, humidity_Max,
humidity_Total, pressure_Min, pressure_Avg, pressure_Max,
pressure_Total, wind_Speed_Min, wind_Speed_Avg, wind_Speed_Max,
wind_Speed_Total, dew_Point_Min, dew_Point_Avg, dew_Point_Max,
dew_Point_Total, heat_Index_Min, heat_Index_Avg, heat_Index_Max,
heat_Index_Total, precipitation_Q1, precipitation_Q2,
precipitation_Q3, precipitation_Q4;
float temperature_Mapped, temperature_Min_Mapped,
temperature_Avg_Mapped, temperature_Max_Mapped, humidity_Mapped,
humidity_Min_Mapped, humidity_Avg_Mapped, humidity_Max_Mapped,
pressure_Mapped, pressure_Min_Mapped, pressure_Avg_Mapped,
pressure_Max_Mapped, wind_Speed_Mapped, wind_Speed_Min_Mapped,
wind_Speed_Avg_Mapped, wind_Speed_Max_Mapped, dew_Point_Mapped,
dew_Point_Min_Mapped, dew_Point_Avg_Mapped, dew_Point_Max_Mapped,
heat_Index_Mapped, heat_Index_Min_Mapped, heat_Index_Avg_Mapped,
heat_Index_Max_Mapped;
long avg_Counter = 0;

#define DATA_REFRESH_RATE 1000
unsigned long pageRefreshTimer = millis();
bool newPageLoaded = false;
```

```
void setup() {

  myNex.begin();
  delay(500);
  myNex.writeStr("page 0");
  delay(50);
  myNex.lastCurrentPageId = 1;

  //    Serial.begin(9600);
  radio.begin();
  radio.openReadingPipe(0, address);
  radio.setPALevel(RF24_PA_MIN);
  radio.startListening();


  io.connect();
  while (io.status() < AIO_CONNECTED)
  {
    Serial.print(".");
    delay(500);
  }


  configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
  dateTime();


  temperature_Min = 1000;
  temperature_Max = 0;
  temperature_Total = 0;
  humidity_Min = 1000;
  humidity_Max = 0;
  humidity_Total = 0;
  pressure_Min = 2500;
  pressure_Max = 0;
  pressure_Total = 0;
  wind_Speed_Min = 1000;
  wind_Speed_Max = 0;
  wind_Speed_Total = 0;
  dew_Point_Min = 1000;
  dew_Point_Max = 0;
  dew_Point_Total = 0;
  heat_Index_Min = 1000;
  heat_Index_Max = 0;
  heat_Index_Total = 0;
  last_Date = atoi(DD);

}

void loop() {

  myNex.NextionListen();

  readSensorValues();

  calculateDetailedWeatherData();

  resetDetailedData();
```

```
  refereshCurrentPage();

  firstRefresh();

  timer = millis();
  if (timer - timer_Save_Data >= 600000) { //10 minutes 600000
    save_Data();
    timer_Save_Data = timer;
  }

  if (timer - timer_Update_Cloud >= 600000) { // max 30 records per
minute
    update_Cloud();
    timer_Update_Cloud = timer;
  }

  io.run();

}

void firstRefresh() {
  if (myNex.currentPageId != myNex.lastCurrentPageId) {
    newPageLoaded = true;

    switch (myNex.currentPageId) {
      case 0:
        refreshPage0();
        break;

      case 1:
        refreshPage1();
        break;

      case 2:
        refreshPage2();
        break;

      case 3:
        refreshPage3();
        break;

      case 4:
        refreshPage4();
        break;

      case 5:
        refreshPage5();
        break;

      case 6:
        refreshPage6();
        break;

      case 7:
        refreshPage7();
        break;
```

```
      case 8:
        refreshPage8();
        break;

      case 9:
        refreshPage9();
        break;

      case 10:
        refreshPage10();
        break;

      case 11:
        refreshPage11();
        break;

      case 12:
        refreshPage12();
        break;

      case 13:
        refreshPage13();
        break;

      case 14:
        refreshPage14();
        break;

      case 15:
        refreshPage15();
        break;
    }

    newPageLoaded = false;
    myNex.lastCurrentPageId = myNex.currentPageId;
  }
}

void readSensorValues() {

  if (radio.available()) {
    radio.read(&Weather_Data_01, sizeof(Weather_Data_01));

    rainfall = Weather_Data_01[0];
    directionID = int(Weather_Data_01[1]);
    wind_Direction = direction_Name[directionID];
    wind_Angle = Weather_Data_01[2];
    wind_Speed = Weather_Data_01[3];
    temperature = Weather_Data_01[4];
    humidity = Weather_Data_01[5];
    heat_Index = Weather_Data_01[6];
    pressure = Weather_Data_01[7];
    dew_Point = temperature - ((100 - humidity) / 5);

    sea_Level_Pressure = pressure * pow(1 - (0.0065 *
altitude_meters) / (temperature + (0.0065 * altitude_meters) +
273.15), -5.257 );
```

```
  }

  dateTime();

  solarTime();

  calcMoonPhase();

}


void refereshCurrentPage() {
  if ((millis() - pageRefreshTimer) > DATA_REFRESH_RATE) {
    switch (myNex.currentPageId) {
      case 0:
        refreshPage0();
        break;

      case 1:
        refreshPage1();
        break;

      case 2:
        refreshPage2();
        break;

      case 3:
        refreshPage3();
        break;

      case 4:
        refreshPage4();
        break;

      case 5:
        refreshPage5();
        break;

      case 6:
        refreshPage6();
        break;

      case 7:
        refreshPage7();
        break;

      case 8:
        refreshPage8();
        break;

      case 9:
        refreshPage9();
        break;

      case 10:
        refreshPage10();
        break;
```

```
      case 11:
        refreshPage11();
        break;

      case 12:
        refreshPage12();
        break;

      case 13:
        refreshPage13();
        break;

      case 14:
        refreshPage14();
        break;

      case 15:
        refreshPage15();
        break;
    }

    pageRefreshTimer = millis();
  }
}

void refreshPage0() {

  String tempString1 = String(temperature, 2);
  myNex.writeStr("t0.txt", tempString1);

  String tempString2 = String(humidity, 2);
  myNex.writeStr("t2.txt", tempString2);

  String tempString3 = String(pressure, 2);
  myNex.writeStr("t4.txt", tempString3);

  String tempString4 = wind_Direction;
  myNex.writeStr("t6.txt", tempString4);

  String tempString5 = String(wind_Angle, 1);
  myNex.writeStr("t7.txt", tempString5);

  String tempString6 = String(wind_Speed, 2);
  myNex.writeStr("t9.txt", tempString6);

  String tempString7 = String(rainfall, 2);
  myNex.writeStr("t11.txt", tempString7);

  String tempString8 = String(dew_Point, 2);
  myNex.writeStr("t13.txt", tempString8);

  String tempString9 = String(heat_Index, 2);
  myNex.writeStr("t15.txt", tempString9);

  String tempString10 = String(timeText);
  myNex.writeStr("t19.txt", tempString10);

  String tempString0 = moonPhase;
```

```
  myNex.writeStr("t17.txt", tempString0);

  String tempString = String(dateText);
  myNex.writeStr("t18.txt", tempString);

  if (timeHour >= 5 && timeHour <= 6) {
    myNex.writeNum("p3.pic", 0);
  } else {
    if (timeHour >= 7 && timeHour <= 9) {
      myNex.writeNum("p3.pic", 4);
    } else {
      if (timeHour >= 10 && timeHour <= 11) {
        myNex.writeNum("p3.pic", 5);
      } else {
        if (timeHour >= 12 && timeHour <= 13) {
          myNex.writeNum("p3.pic", 6);
        } else {
          if (timeHour >= 14 && timeHour <= 15) {
            myNex.writeNum("p3.pic", 7);
          } else {
            if (timeHour >= 16 && timeHour <= 17) {
              myNex.writeNum("p3.pic", 8);
            } else {
              if (timeHour >= 18 && timeHour <= 19) {
                myNex.writeNum("p3.pic", 9);
              } else {
                if (timeHour >= 20 && timeHour <= 23) {
                  myNex.writeNum("p3.pic", 10);
                } else {
                  if (timeHour >= 0 && timeHour <= 4) {
                    myNex.writeNum("p3.pic", 10);
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}

void refreshPage1() {

  String tempString11 = sunriseTxt;
  myNex.writeStr("t0.txt", tempString11);

  String tempString12 = weather_Prediction_Status;
  myNex.writeStr("t1.txt", tempString12);

  String tempString13 = sunsetTxt;
  myNex.writeStr("t2.txt", tempString13);

  String tempString14 = solarnoonTxt;
  myNex.writeStr("t4.txt", tempString14);

  String tempString15 = String(timeText);
  myNex.writeStr("t19.txt", tempString15);
```

```
  String tempString16 = String(dateText);
  myNex.writeStr("t18.txt", tempString16);

  if (timeHour >= 5 && timeHour <= 6) {
    myNex.writeNum("p3.pic", 2);
  } else {
    if (timeHour >= 7 && timeHour <= 9) {
      myNex.writeNum("p3.pic", 12);
    } else {
      if (timeHour >= 10 && timeHour <= 11) {
        myNex.writeNum("p3.pic", 13);
      } else {
        if (timeHour >= 12 && timeHour <= 13) {
          myNex.writeNum("p3.pic", 14);
        } else {
          if (timeHour >= 14 && timeHour <= 15) {
            myNex.writeNum("p3.pic", 15);
          } else {
            if (timeHour >= 16 && timeHour <= 17) {
              myNex.writeNum("p3.pic", 16);
            } else {
              if (timeHour >= 18 && timeHour <= 19) {
                myNex.writeNum("p3.pic", 17);
              } else {
                if (timeHour >= 20 && timeHour <= 23) {
                  myNex.writeNum("p3.pic", 18);
                } else {
                  if (timeHour >= 0 && timeHour <= 4) {
                    myNex.writeNum("p3.pic", 18);
                  }
                }
              }
            }
          }
        }
      }
    }
  }

}

void refreshPage2() {

}

void refreshPage3() {

}

void refreshPage4() {

}

void refreshPage5() {

  String tempString20 = String(timeText);
  myNex.writeStr("t19.txt", tempString20);
```

```
  String tempString21 = String(dateText);
  myNex.writeStr("t18.txt", tempString21);

  String tempString22 = String(temperature, 2);
  myNex.writeStr("t0.txt", tempString22);

  String tempString23 = String(temperature_Min, 2);
  myNex.writeStr("t3.txt", tempString23);

  String tempString24 = String(temperature_Avg, 2);
  myNex.writeStr("t5.txt", tempString24);

  String tempString25 = String(temperature_Max, 2);
  myNex.writeStr("t7.txt", tempString25);

  myNex.writeNum("j0.val", temperature_Mapped);

  myNex.writeNum("j1.val", temperature_Min_Mapped);

  myNex.writeNum("j2.val", temperature_Avg_Mapped);

  myNex.writeNum("j3.val", temperature_Max_Mapped);

}

void refreshPage6() {

  String tempString26 = String(timeText);
  myNex.writeStr("t19.txt", tempString26);

  String tempString27 = String(dateText);
  myNex.writeStr("t18.txt", tempString27);

  String tempString28 = String(humidity, 2);
  myNex.writeStr("t2.txt", tempString28);

  String tempString29 = String(humidity_Min, 2);
  myNex.writeStr("t0.txt", tempString29);

  String tempString30 = String(humidity_Avg, 2);
  myNex.writeStr("t4.txt", tempString30);

  String tempString31 = String(humidity_Max, 2);
  myNex.writeStr("t6.txt", tempString31);

  myNex.writeNum("z0.val", humidity_Mapped);

  myNex.writeNum("z1.val", humidity_Min_Mapped);

  myNex.writeNum("z2.val", humidity_Avg_Mapped);

  myNex.writeNum("z3.val", humidity_Max_Mapped);

}

void refreshPage7() {
```

```
  String tempString32 = String(timeText);
  myNex.writeStr("t19.txt", tempString32);

  String tempString33 = String(dateText);
  myNex.writeStr("t18.txt", tempString33);

  String tempString34 = String(pressure, 2);
  myNex.writeStr("t4.txt", tempString34);

  String tempString35 = String(pressure_Min, 2);
  myNex.writeStr("t0.txt", tempString35);

  String tempString36 = String(pressure_Avg, 2);
  myNex.writeStr("t2.txt", tempString36);

  String tempString37 = String(pressure_Max, 2);
  myNex.writeStr("t6.txt", tempString37);

  myNex.writeNum("z0.val", pressure_Mapped);

  myNex.writeNum("z1.val", pressure_Min_Mapped);

  myNex.writeNum("z2.val", pressure_Avg_Mapped);

  myNex.writeNum("z3.val", pressure_Max_Mapped);

}

void refreshPage8() {

  String tempString38 = String(timeText);
  myNex.writeStr("t19.txt", tempString38);

  String tempString39 = String(dateText);
  myNex.writeStr("t18.txt", tempString39);

  String tempString40 = wind_Direction;
  myNex.writeStr("t6.txt", tempString40);

  String tempString41 = String(wind_Angle, 1);
  myNex.writeStr("t7.txt", tempString41);

  switch (directionID) {
    case 0:
      myNex.writeNum("p1.pic", 63);
      break;

    case 1:
      myNex.writeNum("p1.pic", 75);
      break;

    case 2:
      myNex.writeNum("p1.pic", 62);
      break;

    case 3:
      myNex.writeNum("p1.pic", 70);
      break;
```

```
    case 4:
      myNex.writeNum("p1.pic", 69);
      break;

    case 5:
      myNex.writeNum("p1.pic", 71);
      break;

    case 6:
      myNex.writeNum("p1.pic", 68);
      break;

    case 7:
      myNex.writeNum("p1.pic", 61);
      break;

    case 8:
      myNex.writeNum("p1.pic", 65);
      break;

    case 9:
      myNex.writeNum("p1.pic", 73);
      break;

    case 10:
      myNex.writeNum("p1.pic", 72);
      break;

    case 11:
      myNex.writeNum("p1.pic", 66);
      break;

    case 12:
      myNex.writeNum("p1.pic", 64);
      break;

    case 13:
      myNex.writeNum("p1.pic", 76);
      break;

    case 14:
      myNex.writeNum("p1.pic", 67);
      break;

    case 15:
      myNex.writeNum("p1.pic", 74);
      break;
  }
}

void refreshPage9() {

  String tempString42 = String(timeText);
  myNex.writeStr("t19.txt", tempString42);

  String tempString43 = String(dateText);
  myNex.writeStr("t18.txt", tempString43);
```

```
  String tempString44 = String(wind_Speed, 2);
  myNex.writeStr("t9.txt", tempString44);

  String tempString45 = String(wind_Speed_Min, 2);
  myNex.writeStr("t1.txt", tempString45);

  String tempString46 = String(wind_Speed_Avg, 2);
  myNex.writeStr("t3.txt", tempString46);

  String tempString47 = String(wind_Speed_Max, 2);
  myNex.writeStr("t5.txt", tempString47);

  myNex.writeNum("z0.val", wind_Speed_Mapped);

  myNex.writeNum("z1.val", wind_Speed_Min_Mapped);

  myNex.writeNum("z2.val", wind_Speed_Avg_Mapped);

  myNex.writeNum("z3.val", wind_Speed_Max_Mapped);

}

void refreshPage10() {

  String tempString48 = String(timeText);
  myNex.writeStr("t19.txt", tempString48);

  String tempString49 = String(dateText);
  myNex.writeStr("t18.txt", tempString49);

  String tempString50 = String(rainfall, 2);
  myNex.writeStr("t11.txt", tempString50);

  String tempString51 = String(precipitation_Q1, 2);
  myNex.writeStr("t1.txt", tempString51);

  String tempString52 = String(precipitation_Q2, 2);
  myNex.writeStr("t4.txt", tempString52);

  String tempString53 = String(precipitation_Q3, 2);
  myNex.writeStr("t2.txt", tempString53);

  String tempString54 = String(precipitation_Q4, 2);
  myNex.writeStr("t6.txt", tempString54);

}

void refreshPage11() {

  String tempString55 = String(timeText);
  myNex.writeStr("t19.txt", tempString55);

  String tempString56 = String(dateText);
  myNex.writeStr("t18.txt", tempString56);

  String tempString57 = weather_Prediction_Status;
  myNex.writeStr("t1.txt", tempString57);
```

```
switch (weather_Prediction) {

  case 0:
    myNex.writeNum("p1.pic", 100);
    break;

  case 1:
    myNex.writeNum("p1.pic", 92);
    break;

  case 2:
    myNex.writeNum("p1.pic", 87);
    break;

  case 3:
    myNex.writeNum("p1.pic", 85);
    break;

  case 4:
    myNex.writeNum("p1.pic", 88);
    break;

  case 5:
    myNex.writeNum("p1.pic", 79);
    break;

  case 6:
    myNex.writeNum("p1.pic", 96);
    break;

  case 7:
    myNex.writeNum("p1.pic", 82);
    break;

  case 8:
    myNex.writeNum("p1.pic", 83);
    break;

  case 9:
    myNex.writeNum("p1.pic", 93);
    break;

  case 10:
    myNex.writeNum("p1.pic", 92);
    break;

  case 11:
    myNex.writeNum("p1.pic", 87);
    break;

  case 12:
    myNex.writeNum("p1.pic", 86);
    break;

  case 13:
    myNex.writeNum("p1.pic", 88);
    break;
```

```
case 14:
  myNex.writeNum("p1.pic", 78);
  break;

case 15:
  myNex.writeNum("p1.pic", 91);
  break;

case 16:
  myNex.writeNum("p1.pic", 97);
  break;

case 17:
  myNex.writeNum("p1.pic", 84);
  break;

case 18:
  myNex.writeNum("p1.pic", 93);
  break;

case 19:
  myNex.writeNum("p1.pic", 77);
  break;

case 20:
  myNex.writeNum("p1.pic", 92);
  break;

case 21:
  myNex.writeNum("p1.pic", 87);
  break;

case 22:
  myNex.writeNum("p1.pic", 92);
  break;

case 23:
  myNex.writeNum("p1.pic", 90);
  break;
case 24:
  myNex.writeNum("p1.pic", 89);
  break;

case 25:
  myNex.writeNum("p1.pic", 80);
  break;

case 26:
  myNex.writeNum("p1.pic", 92);
  break;

case 27:
  myNex.writeNum("p1.pic", 81);
  break;

case 28:
  myNex.writeNum("p1.pic", 98);
```

```
      break;

    case 29:
      myNex.writeNum("p1.pic", 95);
      break;

    case 30:
      myNex.writeNum("p1.pic", 94);
      break;

    case 31:
      myNex.writeNum("p1.pic", 99);
      break;

    case 32:
      myNex.writeNum("p1.pic", 77);
      break;

  }

}

void refreshPage12() {

  String tempString58 = String(timeText);
  myNex.writeStr("t19.txt", tempString58);

  String tempString59 = String(dateText);
  myNex.writeStr("t18.txt", tempString59);

  String tempString60 = String(dew_Point, 2);
  myNex.writeStr("t0.txt", tempString60);

  String tempString61 = String(dew_Point_Min, 2);
  myNex.writeStr("t3.txt", tempString61);

  String tempString62 = String(dew_Point_Avg, 2);
  myNex.writeStr("t5.txt", tempString62);

  String tempString63 = String(dew_Point_Max, 2);
  myNex.writeStr("t7.txt", tempString63);

  myNex.writeNum("j0.val", dew_Point_Mapped);

  myNex.writeNum("j1.val", dew_Point_Min_Mapped);

  myNex.writeNum("j2.val", dew_Point_Avg_Mapped);

  myNex.writeNum("j3.val", dew_Point_Max_Mapped);

}

void refreshPage13() {

  String tempString64 = String(timeText);
  myNex.writeStr("t19.txt", tempString64);

  String tempString65 = String(dateText);
```

```
  myNex.writeStr("t18.txt", tempString65);

  String tempString66 = String(heat_Index, 2);
  myNex.writeStr("t0.txt", tempString66);

  String tempString67 = String(heat_Index_Min, 2);
  myNex.writeStr("t3.txt", tempString67);

  String tempString68 = String(heat_Index_Avg, 2);
  myNex.writeStr("t5.txt", tempString68);

  String tempString69 = String(heat_Index_Max, 2);
  myNex.writeStr("t7.txt", tempString69);

  myNex.writeNum("j0.val", heat_Index_Mapped);

  myNex.writeNum("j1.val", heat_Index_Min_Mapped);

  myNex.writeNum("j2.val", heat_Index_Avg_Mapped);

  myNex.writeNum("j3.val", heat_Index_Max_Mapped);

}

void refreshPage14() {

  String tempString70 = String(timeText);
  myNex.writeStr("t19.txt", tempString70);

  String tempString71 = String(dateText);
  myNex.writeStr("t18.txt", tempString71);

  String tempString72 = moonPhase;
  myNex.writeStr("t17.txt", tempString72);

  switch (mp) {
    case 0:
      myNex.writeNum("p1.pic", 56);
      break;

    case 1:
      myNex.writeNum("p1.pic", 51);
      break;

    case 2:
      myNex.writeNum("p1.pic", 57);
      break;

    case 3:
      myNex.writeNum("p1.pic", 58);
      break;

    case 4:
      myNex.writeNum("p1.pic", 52);
      break;

    case 5:
      myNex.writeNum("p1.pic", 53);
```

```
      break;

    case 6:
      myNex.writeNum("p1.pic", 54);
      break;

    case 7:
      myNex.writeNum("p1.pic", 55);
      break;
  }

}

void refreshPage15() {

  String tempString73 = String(timeText);
  myNex.writeStr("t19.txt", tempString73);

  String tempString74 = String(dateText);
  myNex.writeStr("t18.txt", tempString74);

  String tempString75 = sunriseTxt;
  myNex.writeStr("t0.txt", tempString75);

  String tempString76 = sunsetTxt;
  myNex.writeStr("t2.txt", tempString76);

  String tempString77 = solarnoonTxt;
  myNex.writeStr("t4.txt", tempString77);

}


void dateTime() {

  struct tm timeinfo;
  if (!getLocalTime(&timeinfo)) {
    Serial.println("Failed to obtain time");
    return;
  }

  //  char DD[3];
  strftime(DD, 3, "%d", &timeinfo);

  //  char MMM[15];
  strftime(MMM, 15, "%B", &timeinfo);

  //  char MM[3];
  strftime(MM, 3, "%m", &timeinfo);

  //  char YY[5];
  strftime(YY, 5, "%Y", &timeinfo);

  //  char HH[3];
  strftime(HH, 3, "%I", &timeinfo);

  //  char HH24[3];
  strftime(HH24, 3, "%H", &timeinfo);
```

```
  //  char Mn[3];
  strftime(Mn, 3, "%M", &timeinfo);

  //  char sec[3];
  strftime(sec, 3, "%S", &timeinfo);

  //  char half[3];
  strftime(half, 3, "%p", &timeinfo);

  dateText = String(DD) + " " + String(MMM) + " " + String(YY);
  timeText = String(HH) + ":" + String(Mn) + ":" + String(sec) + " "
+ String(half);
  //  String tHour = String(HH24);
  //  String tHr = tHour.substring(0,2);
  //  timeHour = tHr.toInt();
  timeHour = atoi(HH24);

}

void solarTime() {

  D = atoi(DD);
  M = atoi(MM);
  Y = atoi(YY);

  int homeSunrise  = Home.sunrise(Y, M, D, false);
  int homeSunset   = Home.sunset(Y, M, D, false);

  homeSunrise = homeSunrise + 30;
  homeSunset = homeSunset + 30;

  char time1[6];
  Dusk2Dawn::min2str(time1, homeSunrise);

  char time2[6];
  Dusk2Dawn::min2str(time2, homeSunset);

  int homeSolarNoon = homeSunrise + (homeSunset - homeSunrise) / 2;
  char time3[6];

  Dusk2Dawn::min2str(time3, homeSolarNoon);
  //  char time4[6];

  sunriseTxt = String(time1);
  sunsetTxt = String(time2);
  solarnoonTxt = String(time3);

}

void calcMoonPhase() {
  mp = moon_phase_val();

  switch (mp) {
    case 0:
      moonPhase = "Full Moon";
      break;
```

```
    case 1:
      moonPhase = "Waning Gibbous";
      break;

    case 2:
      moonPhase = "Last Quarter";
      break;

    case 3:
      moonPhase = "Old Crescent";
      break;

    case 4:
      moonPhase = "New Moon";
      break;

    case 5:
      moonPhase = "New Crescent";
      break;

    case 6:
      moonPhase = "First Quarter";
      break;

    case 7:
      moonPhase = "Waxing Gibbous";
      break;
  }
}

int moon_phase_val() {
  double julian_Date = 0;
  double elapsed_Date = 0;
  int valu = 0;

  julian_Date = julianDate(Y, M, D);
  julian_Date = int(julian_Date - 2244116.75);
  julian_Date /= 29.53;
  valu = julian_Date;
  julian_Date -= valu;
  elapsed_Date = julian_Date * 29.53;

  valu = julian_Date * 8 + 0.5;
  valu = valu & 7;
  return valu;

}

double julianDate(int y, int m, int d) {

  int mnth, yer;
  double key1, key2, key3;
  double julian_Output;

  yer = y - int((12 - m) / 10);
  mnth = m + 9;
  if (mnth >= 12) {
    mnth = mnth - 12;
```

```
  }
  key1 = 365.25 * (yer + 4172);
  key2 = int((30.6001 * mnth) + 0.5);
  key3 = int((((yer / 100) + 4) * 0.75) - 38);
  julian_Output = key1 + key2 + d + 59;
  julian_Output = julian_Output - key3;

  return julian_Output;
}


void save_Data() {
  if (counter == 18) {
    counter = 0;
  } else {
    pressureArray[counter] = sea_Level_Pressure;
    counter++;
  }

  current_Pressure = (pressureArray[17] + pressureArray[16] +
pressureArray[15] + pressureArray[14] + pressureArray[13]) / 5;
  previous_Pressure = (pressureArray[0] + pressureArray[1] +
pressureArray[2] + pressureArray[3] + pressureArray[4]) / 5;

  if ((current_Pressure - previous_Pressure) >= 1.6 &&
sea_Level_Pressure >= 947 && sea_Level_Pressure <= 1030) {
    trend = "Rising";
  } else {
    if ((previous_Pressure - current_Pressure) >= 1.6 &&
sea_Level_Pressure >= 960 && sea_Level_Pressure <= 1033) {
      trend = "Falling";
    }  else {
      if ((current_Pressure - previous_Pressure) <= 1.6 ||
(previous_Pressure - current_Pressure) <= 1.6 && sea_Level_Pressure
>= 947 && sea_Level_Pressure <= 1030) {
        trend = "Steady";
      } else {
        trend = "Error";
      }
    }
  }

  get_Zambretti_Value();
}

void get_Zambretti_Value() {
  if (trend == "Falling") {
    zambretti_Calc = 127 - 0.12 * sea_Level_Pressure;
  } else {
    if (trend == "Steady") {
      zambretti_Calc = 144 - 0.13 * sea_Level_Pressure;
    } else {
      if (trend == "Rising") {
        zambretti_Calc = 185 - 0.16 * sea_Level_Pressure;
      } else {
        weather_Prediction_Status = weather[0];
      }
    }
```

```
  }
  weather_Prediction = round(zambretti_Calc);
  add_Wind_Effect();
}

void add_Wind_Effect() {
  if (wind_Angle >= 46.0 && wind_Angle <= 135.0) {
    weather_Prediction = weather_Prediction + 1;
  } else {
    if (wind_Angle >= 136.0 && wind_Angle <= 225.0) {
      weather_Prediction = weather_Prediction + 2;
    } else {
      if (wind_Angle >= 226.0 && wind_Angle <= 315.0) {
        weather_Prediction = weather_Prediction + 1;
      } else {
        weather_Prediction = weather_Prediction + 0;
      }
    }
  }
  weather_Prediction_Status = weather[weather_Prediction];
}

void calculateDetailedWeatherData() {

  avg_Counter++;

  temperature_Mapped = map(temperature, 0, 50, 17, 88);

  //Minimum Temperature
  if (temperature <= temperature_Min ) {
    temperature_Min = temperature;
  } else {
    temperature_Min = temperature_Min;
  }

  temperature_Min_Mapped = map(temperature_Min, 0, 50, 17, 88);

  //Average Temperature
  temperature_Total = temperature_Total + temperature;
  temperature_Avg = temperature_Total / avg_Counter;

  temperature_Avg_Mapped = map(temperature_Avg, 0, 50, 17, 88);

  //Maximum Temperature
  if (temperature >= temperature_Max) {
    temperature_Max = temperature;
  } else {
    temperature_Max = temperature_Max;
  }

  temperature_Max_Mapped = map(temperature_Max, 0, 50, 17, 88);


  humidity_Mapped = map(humidity, 0, 100, 0, 298);

  //Minimum Humidity
  if (humidity <= humidity_Min) {
    humidity_Min = humidity;
```

```
} else {
  humidity_Min = humidity_Min;
}

humidity_Min_Mapped = map(humidity_Min, 0, 100, 0, 298);

//Average Humidity
humidity_Total = humidity_Total + humidity;
humidity_Avg = humidity_Total / avg_Counter;

humidity_Avg_Mapped = map(humidity_Avg, 0, 100, 0, 298);

//Maximum Humidity
if (humidity >= humidity_Max ) {
  humidity_Max = humidity;
} else {
  humidity_Max = humidity_Max;
}

humidity_Max_Mapped = map(humidity_Max, 0, 100, 0, 298);

pressure_Mapped = map(pressure, 950, 1050, 0, 165);

//Minimum Pressure
if (pressure <= pressure_Min) {
  pressure_Min = pressure;
} else {
  pressure_Min = pressure_Min;
}

pressure_Min_Mapped = map(pressure_Min, 950, 1050, 0, 165);

//Average Pressure
pressure_Total = pressure_Total + pressure;
pressure_Avg = pressure_Total / avg_Counter;

pressure_Avg_Mapped = map(pressure_Avg, 950, 1050, 0, 165);

//Maximum Pressure
if (pressure >= pressure_Max) {
  pressure_Max = pressure;
} else {
  pressure_Max = pressure_Max;
}

pressure_Max_Mapped = map(pressure_Max, 950, 1050, 0, 165);

wind_Speed_Mapped = map(wind_Speed, 0, 60, 0, 300);

//Minimum Wind Speed
if (wind_Speed <= wind_Speed_Min) {
  wind_Speed_Min = wind_Speed;
} else {
  wind_Speed_Min = wind_Speed_Min;
}

wind_Speed_Min_Mapped = map(wind_Speed_Min, 0, 60, 0, 300);
```

```
//Average Wind Speed
wind_Speed_Total = wind_Speed_Total + wind_Speed;
wind_Speed_Avg = wind_Speed_Total / avg_Counter;

wind_Speed_Avg_Mapped = map(wind_Speed_Avg, 0, 60, 0, 300);

//Maximum Wind Speed
if (wind_Speed >= wind_Speed_Max) {
  wind_Speed_Max = wind_Speed;
} else {
  wind_Speed_Max = wind_Speed_Max;
}

wind_Speed_Max_Mapped = map(wind_Speed_Max, 0, 60, 0, 300);

dew_Point_Mapped = map(dew_Point, 0, 50, 17, 88);

//Minimum Dew Point
if (dew_Point <= dew_Point_Min) {
  dew_Point_Min = dew_Point;
} else {
  dew_Point_Min = dew_Point_Min;
}

dew_Point_Min_Mapped = map(dew_Point_Min, 0, 50, 17, 88);

//Average Dew Point
dew_Point_Total = dew_Point_Total + dew_Point;
dew_Point_Avg = dew_Point_Total / avg_Counter;

dew_Point_Avg_Mapped = map(dew_Point_Avg, 0, 50, 17, 88);

//Maximum Dew Point
if (dew_Point >= dew_Point_Max) {
  dew_Point_Max = dew_Point;
} else {
  dew_Point_Max = dew_Point_Max;
}

dew_Point_Max_Mapped = map(dew_Point_Max, 0, 50, 17, 88);


heat_Index_Mapped = map(heat_Index, 0, 50, 17, 88);

//Minimum Heat Index
if (heat_Index <= heat_Index_Min) {
  heat_Index_Min = heat_Index;
} else {
  heat_Index_Min = heat_Index_Min;
}

heat_Index_Min_Mapped = map(heat_Index_Min, 0, 50, 17, 88);

//Average Heat Index
heat_Index_Total = heat_Index_Total + heat_Index;
heat_Index_Avg = heat_Index_Total / avg_Counter;

heat_Index_Avg_Mapped = map(heat_Index_Avg, 0, 50, 17, 88);
```

```
  //Maximum Heat Index
  if (heat_Index >= heat_Index_Max) {
    heat_Index_Max = heat_Index;
  } else {
    heat_Index_Max = heat_Index_Max;
  }

  heat_Index_Max_Mapped = map(heat_Index_Max, 0, 50, 17, 88);

  quartallyPrecipitation();

  resetDetailedData();

}


void quartallyPrecipitation() {

  if (timeHour >= 0 && timeHour <= 5) {
    precipitation_Q1 = rainfall;
    precipitation_Q2 = 0.0;
    precipitation_Q3 = 0.0;
    precipitation_Q4 = 0.0;
  } else {
    if (timeHour >= 6 && timeHour <= 11) {
      precipitation_Q1 = precipitation_Q1;
      precipitation_Q2 = rainfall - precipitation_Q1;
      precipitation_Q3 = 0.0;
      precipitation_Q4 = 0.0;
    } else {
      if (timeHour >= 12 && timeHour <= 17) {
        precipitation_Q1 = precipitation_Q1;
        precipitation_Q2 = precipitation_Q2;
        precipitation_Q3 = rainfall - (precipitation_Q1 +
precipitation_Q2);
        precipitation_Q4 = 0.0;
      } else {
        if (timeHour >= 18 && timeHour <= 23) {
          precipitation_Q1 = precipitation_Q1;
          precipitation_Q2 = precipitation_Q2;
          precipitation_Q3 = precipitation_Q3;
          precipitation_Q4 = rainfall - (precipitation_Q1 +
precipitation_Q2 + precipitation_Q3);
        }
      }
    }
  }
}


void resetDetailedData() {

  if (atoi(DD) != last_Date) {
    //list of variable need to be reset
    avg_Counter = 0;
    temperature_Min = 1000;
    temperature_Avg = 0;
```

```
    temperature_Max = 0;
    temperature_Total = 0;
    humidity_Min = 1000;
    humidity_Avg = 0;
    humidity_Max = 0;
    humidity_Total = 0;
    pressure_Min = 2500;
    pressure_Avg = 0;
    pressure_Max = 0;
    pressure_Total = 0;
    wind_Speed_Min = 1000;
    wind_Speed_Avg = 0;
    wind_Speed_Max = 0;
    wind_Speed_Total = 0;
    dew_Point_Min = 1000;
    dew_Point_Avg = 0;
    dew_Point_Max = 0;
    dew_Point_Total = 0;
    heat_Index_Min = 1000;
    heat_Index_Avg = 0;
    heat_Index_Max = 0;
    heat_Index_Total = 0;
    precipitation_Q1 = 0;
    precipitation_Q2 = 0;
    precipitation_Q3 = 0;
    precipitation_Q4 = 0;

    last_Date = atoi(DD);
  }
}

void update_Cloud() {
  temperature_val->save(temperature);
  humidity_val->save(humidity);
  pressure_val->save(pressure);
  wind_direction_val->save(wind_Direction);
  wind_speed_val->save(wind_Speed);
  precipitation_val->save(String(rainfall, 2) + " mm");
  dew_point_val->save(dew_Point);
  heat_index_val->save(heat_Index);
  weather_prediction_val->save(weather_Prediction_Status);
}
```