

**DEPARTMENT OF PHYSICS
UNIVERSITY OF COLOMBO**

IoT with Raspberry Pi B3

Name: D.H.C.I.Dharmarathne

Index: s14885

Date: 25/07/2023

ABSTRACT

The objective of this practical was to create a Raspberry Pi-based Internet of Things (IoT) project that involved data collection from a DHT-11 sensor (temperature and humidity), controlling a relay based on humidity levels, and transferring the data to a PC using both local database storage and MQTT protocol. The primary goal was to demonstrate a complete end-to-end solution for data monitoring and transfer in a home automation.

In this IoT project, a Raspberry Pi was utilized to control an LED through GPIO, obtain temperature and humidity data from a DHT-11 sensor, and automate a relay based on humidity levels using a transistor as a switch. The collected data was displayed in the terminal and stored in a local PostgreSQL database on the Raspberry Pi. Additionally, the data was transmitted to a remote PC and uploaded into its PostgreSQL database using the IP address for communication. Moreover, MQTT was employed to facilitate data transfer between the Raspberry Pi and the PC, enabling seamless real-time monitoring and control.

The practical successfully demonstrated the feasibility of building a comprehensive IoT system using a Raspberry Pi. By integrating various components and technologies, including sensor data acquisition, database storage, and MQTT communication, the project showcased the Raspberry Pi's versatility in home automation and data monitoring applications. Despite some challenges in sensor accuracy, data transfer reliability, and relay control precision, the project's overall results were promising. To enhance the system's robustness and accuracy, future improvements could focus on calibrating the sensor, optimizing MQTT configurations, and refining the hardware setup for the relay control. With these enhancements, the Raspberry Pi-based IoT project can be further leveraged for practical applications in smart homes, environmental monitoring, and other data-driven automation scenarios.

TABLE OF CONTENTS

1	INTRODUCTION.....	1
1.1	Raspberry Pi B3.....	1
1.2	PostgreSQL.....	1
1.3	Mosquitto MQTT Broker	2
1.4	DHT 11 Humidity & Temperature Sensor	2
1.5	Relay module.....	2
1.6	Transistor Switch.....	2
1.7	Paho MQTT Client Library	2
2	THEORY.....	3
2.1	Raspberry Pi B3.....	3
2.1.1	Raspberry Pi B3 Pin diagram	4
2.1.2	Raspberry Pi B3 Communication Protocols.....	6
2.1.3	Raspberry pi Serial Communication	7
2.2	DHT-11 Humidity & Temperature Sensor	8
2.2.1	DHT-11 Sensor pin configure	9
2.3	5V Relay Module.....	9
2.3.1	5V Relay module pin configuration	10
2.4	Transistor Switch.....	11
2.4.1	Cut Off State (Open Switch)	11
2.4.2	Saturation State (Closed Switch).....	12
3	METHODOLOGY	13
3.1	Setting up Raspberry Pi B3	13
3.2	Updating Raspberry Pi OS	17
3.3	Blinking LED Using Python.....	18
3.3.1	Creating the circuit for blinking LED	18
3.3.2	Code for blinking LED from Raspberry Pi	19
3.4	Setting up the Circuit & getting the data from DHT-11	19
3.4.1	Creating the circuit for getting the data from DHT-11	19
3.4.2	Import Adafruit_DHT library.....	20
3.4.3	Code for getting data from DHT-11 sensor	21

3.5	Setting up the circuit and Control Relay using DHT-11 Humidity data	22
3.5.1	Creating a circuit for getting data from DHT-11 and Controlling Relay	22
3.5.2	Code for getting the data from DHT-11 and Controlling the Relay	23
3.6	Installing the PostgreSQL Database in Raspberry Pi	24
3.7	Uploading Sensor data into PostgreSQ in Raspberry PI	25
3.7.1	Creating new database in PostgreSQL database	25
3.7.2	Code for uploading the sensor data into database	26
3.8	Installing the PostgreSQ in PC	28
3.9	Uploading Sensor data into PostgreSQ in PC	34
3.9.1	Creating new database in PostgreSQL database	34
3.9.2	Find the IP address in Local network.	36
3.9.3	Code for uploading the sensor data into database	37
3.10	Installing the Mosquitto MQTT server in Raspberry Pi	38
3.11	Uploading the sensor data into database using MQTT service.....	39
3.11.1	Publishing data from Raspberry Pi to Mqtt Client.....	39
3.11.2	Receiving and Uploading data from Mqtt client to Database	41
4	RESULTS AND ANALYSIS	43
4.1	Raspberry Pi Desktop view	43
4.2	The view of LED blinking.....	43
4.3	Getting data from DHT-11 and Printing in the Terminal.....	44
4.4	Uploading the data into local database in Raspberry Pi	44
4.5	Uploading the data into PC database using IP address.....	45
4.6	Uploading the data into PC database using MQTT client.....	46
5	DISCUSSION	49
6	CONCLUSION	50
7	REFERENCES.....	51

TABLE OF FIGURES

Figure 2- 1: Figure of Raspberry pi B3 module (Source: www.raspberrypi.com/news/raspberrypi-b3/)	3
---	---

Figure 2- 2: Figure of Raspberry pi B3 pin diagram (Source: www.raspberrypi.com/documentation/computer/raspberry-pi.html).....	4
Figure 2- 3: Figure of typical application for MCU with DHT-11 sensor Connection (Electronics, 2022)	8
Figure 2- 4: Figure of pin configuration of DHT-11 Sensor.....	9
Figure 2- 5: Figure of pinout of 5V relay module (Barik, 2023)	10
Figure 2- 6: Circuit diagram of Transistor as Open Switch mode (Saini, 2021)	11
Figure 2- 7: Circuit diagram of Transistor as Close Switch mode (Saini, 2021).....	12
Figure 3- 1: Figure of Download page of Raspberry Pi OS and Imager.....	13
Figure 3- 2: Figure of Raspberry Pi Imager Software.....	13
Figure 3- 3: Figure of choosing the file location of Raspberry Pi OS	14
Figure 3- 4: Figure of choosing the microSD card for Installing OS.....	15
Figure 3- 5: Figure of after choosing the OS and SD card locations.	15
Figure 3- 6: Figure of adjusting the settings of OS	16
Figure 3- 7: (a) Figure of showing the Terminal of Raspberry Pi, (b) Figure of after opening the Terminal in Raspberry Pi	17
Figure 3- 8: Circuit diagram of connecting the LED with Raspberry Pi	18
Figure 3- 9: Circuit diagram connecting the DHT-11 with Raspberry Pi.....	20
Figure 3- 10: Circuit diagram for connecting DHT-11 & Relay with Raspberry Pi.....	22
Figure 3- 11: Download page of PostgreSQL.....	28
Figure 3- 12: figure of first installing step of PostgreSQL	29
Figure 3- 13: figure of second installing step of PostgreSQL.....	29
Figure 3- 14: figure of third installing step of PostgreSQL	30
Figure 3- 15: figure of fourth installing step of PostgreSQL	31
Figure 3- 16: figure of fifth installing step of PostgreSQL.....	31
Figure 3- 17: figure of sixth installing step of PostgreSQL	32
Figure 3- 18: Figure of unlocking the PostgreSQL in PgAdmin	33
Figure 3- 19: Figure of creating new database in PostgreSQL	34
Figure 3- 20: Figure of adding new name for database in PostgreSQL.....	35
Figure 3- 21: Figure of showing the IP address in Command Prompt.....	36
Figure 4-1: Figure of Raspberry Pi Desktop View	43
Figure 4-2: (a) Figure of ON state of LED, (b) Figure of OFF state of LED	43
Figure 4-3: Figure of getting data from DHT-11 and printing data	44
Figure 4-4:(a) Figure of printing the sensor data in terminal, (b) Figure of the local database view in Raspberry Pi	45
Figure 4-5: Figure of resulting view of IP addresses	45
Figure 4-6:(a) Figure of PgAdmin database data view in PC	46
Figure 4-7: Figure of uploading data from Raspberry Pi to MQTT client.....	47
Figure 4-8: (a) Figure of local database in PC, (b) Figure of receiving data from MQTT client .	47
Figure 4-9: Figure of PgAdmin database data view in PC.....	48

1 INTRODUCTION

The Internet of Things (IoT) is a new concept that is reshaping technology and the way we interact with the world around us. It refers to a network of connected physical objects, devices, and sensors that enable data collection and sharing over the Internet. This connectivity enables seamless communication and automation, bridging the gap between the digital and physical realms. IoT applications cover various fields such as smart homes, healthcare, transportation, agriculture and industrial automation. Using real-time data and analytics, IoT promises to usher in a new era of connected and intelligent devices, increasing efficiency, convenience and effectiveness. At the same time, it also brings privacy, security, and interoperability issues that require careful attention to integrate successfully into our daily lives.

There are several types of IoT modules used in industry. Those are Arduino, ESP32, ESP8266, Raspberry pi, Orange pi etc. Among them, the Raspberry Pi is used for the automation of most systems. In additionally there are several databases for storing sensor data. Those are PostgreSQL, Mysql, MongoDB etc. The PostgreSQL is used for storing the sensor data of automated most systems in the industry. The sensor data is saved using various types of communication protocols. There are a few types of communication protocols for each device. Those are Wi-Fi, Bluetooth, MQTT, LoRa, ZigBee etc.

1.1 Raspberry Pi B3

The Raspberry Pi 3 Model B is a versatile and widely popular single-board computer that has revolutionized the world of hobby electronics and education. Developed by the Raspberry Pi Foundation, this credit card-sized computer was released in 2016 as an upgraded version of its predecessors. The Raspberry Pi 3B has a powerful quad-core ARM Cortex-A53 processor, 1GB of RAM and built-in Wi-Fi and Bluetooth connectivity.

Its GPIO (General Purpose Input/Output) pins allow easy interfacing with a wide range of electronic devices and sensors, making it an ideal platform for DIY projects, prototyping and programming, and electronics learning. The Raspberry Pi 3B supports a variety of operating systems, including the official Raspberry Pi OS (formerly Raspbian) and various Linux distributions, further enhancing its flexibility and usefulness in a variety of applications.

1.2 PostgreSQL

PostgreSQL is a powerful, open-source relational database management system (RDBMS). Developed at the University of California, Berkeley in the 1980s, PostgreSQL has evolved into a feature-rich and robust database solution that rivals commercial alternatives. PostgreSQL boasts advanced features, including support for complex queries, data integrity through constraints, and multi-version concurrency control (MVCC) for efficient handling of multiple concurrent transactions. It also supports various indexing techniques and optimization strategies, resulting in excellent performance and scalability. (aiven.io, 2022)

1.3 Mosquitto MQTT Broker

Eclipse Mosquitto is an open-source (EPL/EDL licensed) message broker that implements the MQTT protocol versions 5.0, 3.1.1 and 3.1. Mosquitto is lightweight and is suitable for use on all devices from low power single board computers to full servers. The MQTT protocol provides a lightweight method of carrying out messaging using a publish/subscribe model. This makes it suitable for Internet of Things messaging such as with low-power sensors or mobile devices such as phones, embedded computers or microcontrollers. The Mosquitto project also provides a C library for implementing MQTT clients, and the very popular `mosquitto_pub` and `mosquitto_sub` command line MQTT clients. (mosquitto, 2022)

1.4 DHT 11 Humidity & Temperature Sensor

The *DHT-11* is a basic, ultra-low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air, and spits out a digital signal on the data pin (no analog input pins are needed). It's fairly simple to use but requires careful timing to grab data. The new data can get from it once every 2 seconds. *DHT-11* sensor provides humidity value in percentage in relative humidity (20 to 90% RH) and temperature values in degrees Celsius (0 to 50 °C) DHT11 sensor uses resistive humidity measurement component, and *NTC* temperature measurement component. (Electronics, 2022)

1.5 Relay module

A relay module is an electronic switching device that allows low-power microcontrollers or microprocessors to control higher-power circuits or devices. It is widely used in various electronic projects and automation systems due to its ability to isolate and protect sensitive electronic components from high voltages and currents. The relay module operates on a 5-volt power supply, which makes it compatible with most common microcontroller boards like Arduino and Raspberry Pi.

1.6 Transistor Switch

A transistor is a miniature semiconductor that regulates or controls current or voltage flow in addition amplifying and generating these electrical signals and acting as a switch/gate for them. Typically, transistors consist of three layers, or terminals, of a semiconductor material, each of which can carry a current. (Awati, 2023)

The transistor operates as a Single Pole Single Throw (SPST) solid state switch. When a zero input signal applied to the base of the transistor, it acts as an open switch. If a positive signal applied at the input terminal then it acts like a closed switch. (Saini, 2021)

1.7 Paho MQTT Client Library

The Paho MQTT Client library is developed and maintained by the Eclipse Paho project, an open-source initiative that aims to provide MQTT implementations for multiple programming languages and platforms. The Paho MQTT Client is available for various programming languages, including Python, Java, C/C++, JavaScript, and more.

2 THEORY

2.1 Raspberry Pi B3

Raspberry Pi is a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation in association with Broadcom. The Raspberry Pi project originally leaned toward the promotion of teaching basic computer science in schools.



Figure 2- 1: Figure of Raspberry pi B3 module (Source: www.raspberrypi.com/news/raspberry-pi-b3/)

- Power : 5 V, 3 A
- Storage : MicroSDXC slot, USB mass Storage device for booting
- Operating Systems : Linux, FreeBSD, NetBSD, OpenBSD, Plan 9, RISC OS, Windows 10 ARM64
- Raspberry Pi has two methods to define output pins. The first one is to define the pin number with the GPIO number. The second is to define a pin number as a normal pin number.

```
1 DHT_PIN = 17
2 GPIO.setup(8, GPIO.OUT)
```

The first method is defined by line number 1. The second method is defined by line number 2.

2.1.1 Raspberry Pi B3 Pin diagram

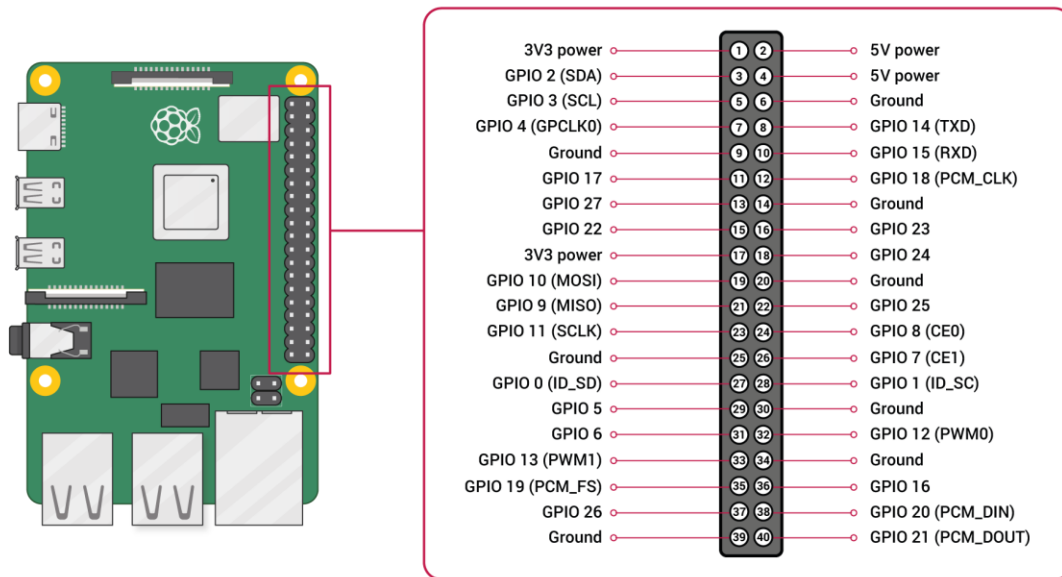


Figure 2- 2: Figure of Raspberry pi B3 pin diagram (Source: www.raspberrypi.com/documentation/computer/raspberry-pi.html)

Raspberry pi 3 Power pins

The board consists of two 5V pins, two 3V3 pins, and 9 ground pins (0V), which are unconfigurable.

5V: The 5v pins directly deliver the 5v supply coming from the mains adaptor. This pin can use to power up the Raspberry Pi, and it can also use to power up other 5v devices.

3.3V: The 3v pin is there to offer a stable 3.3v supply to power components and to test LEDs.

GND: Ground is commonly referred to as GND. All the voltages are measured with respect to the GND voltage.

Raspberry pi 3 Input/Output pins

A GPIO pin that is set as an input will allow a signal to be received by the Raspberry Pi that is sent by a device connected to this pin. A voltage between 1.8V and 3.3V will be read by the Raspberry Pi as HIGH and if the voltage is lower than 1.8V will be read as LOW.

A GPIO pin set as an output pin sends the voltage signal as high (3.3V) or low (0V). When this pin is set to HIGH, the voltage at the output is 3.3V and when set to LOW, the output voltage is 0V.

Along with the simple function of input and output pins, the GPIO pins can also perform a variety of alternative functions. Some specific pins are:

1. PWM (Pulse-width modulation) pins:

- Software PWM is available on all pins
- Hardware PWM is available on these pins only: GPIO12, GPIO13, GPIO18, GPIO19

2. SPI pins:

SPI (Serial Peripheral Interface) is another protocol used for master-slave communication. It is used by the Raspberry pi board to quickly communicate between one or more peripheral devices. Data is synchronized using a clock (SCLK at GPIO11) from the master (RPI) and the data is sent from the Pi to our SPI device using the MOSI (Master Out Slave In) pin. If the SPI device needs to communicate back to Raspberry Pi, then it will send data back using the MISO (Master In Slave Out) pin. There are 5 pins involved in SPI communication:

- GND: Connect all GND pins from all the slave components and the Raspberry Pi 3 board together.
- SCLK: Clock of the SPI. Connect all SCLK pins together.
- MOSI: It stands for Master Out Slave In. This pin is used to send data from the master to a slave.
- MISO: It stands for Master In Slave Out. This pin is used to receive data from a slave to the master.
- CE: It stands for Chip Enable. We need to connect one CE pin per slave (or peripheral devices) in our circuit. By default, we have two CE pins but we can configure more CE pins from the other available GPIO pins.

SPI pins on board:

- SPI0: GPIO9 (MISO), GPIO10 (MOSI), GPIO11 (SCLK), GPIO8 (CE0), GPIO7 (CE1)
- SPI1: GPIO19 (MISO), GPIO20 (MOSI), GPIO21 (SCLK), GPIO18 (CE0), GPIO17 (CE1), GPIO16 (CE2)

3. I2C pins:

I2C is used by the Raspberry Pi board to communicate with devices that are compatible with Inter-Integrated Circuit (a low-speed two-wire serial communication protocol). This communication standard requires master-slave roles between both devices. I2C has two connections: SDA (Serial Data) and SCL (Serial Clock). They work by sending data to and using the SDA connection, and the speed of data transfer is controlled via the SCL pin.

- Data: (GPIO2), Clock (GPIO3)
- EEPROM Data: (GPIO0), EEPROM Clock (GPIO1)

4. UART pins:

Serial communication or the UART (Universal Asynchronous Receiver / Transmitter) pins provide a way to communicate between two microcontrollers or the computers. TX pin is used to transmit the serial data and RX pin is used to receive serial data coming from a different serial device.

- TX (GPIO14)
- RX (GPIO15)

2.1.2 Raspberry Pi B3 Communication Protocols

There are many different communication protocols that can be used to send and receive data to and from a Raspberry Pi. WiFi, Bluetooth, Zigbee, LoRa, MQTT, NFC are some of the most popular protocols. Each protocol has its own advantages and disadvantages, so it's important to choose the right one for your specific project. With the right communication protocol, we will be able to send and receive data to and from your Raspberry Pi with ease.

- **WiFi:** WiFi is one of the most common communication protocols used in IoT projects. The Raspberry Pi has built-in WiFi support, so it can easily connect to a WiFi network. Once connected, you can use Python or another programming language to send and receive data over the network.
- **Bluetooth:** Another popular communication protocol for IoT projects is Bluetooth. The Raspberry Pi has built-in Bluetooth support, so it can easily connect to other devices that support Bluetooth. Once connected, you can use Python or another programming language to send and receive data over the Bluetooth connection.
- **Zigbee:** Zigbee is a wireless communication protocol that is commonly used in IoT projects. It's designed to be low-power and low-cost, making it a great option for IoT projects. To use Zigbee with a Raspberry Pi, you'll need to use a Zigbee dongle and a library like PyZigbee.
- **LoRa:** LoRa (Long Range) is a wireless communication protocol that is designed for long-range communication. It's a great option for IoT projects that need to send data over long distances. To use LoRa with a Raspberry Pi, you'll need to use a LoRa dongle and a library like PyLoRa.
- **MQTT:** MQTT (Message Queuing Telemetry Transport) is a lightweight communication protocol that is designed for IoT projects. It's a great option for sending and receiving data in real-time. To use MQTT with a Raspberry Pi, you'll need to use a library like Mosquitto and a MQTT broker.

- **NFC:** NFC (Near Field Communication) is a wireless communication protocol that is designed for short-range communication. It's a great option for IoT projects that need to send data over short distances. To use NFC with a Raspberry Pi, you'll need to use a NFC reader/writer and a library like libnfc.

2.1.3 Raspberry pi Serial Communication

Serial communication refers to the sequential transfer of data over a wire. Data communication between two devices can be wired or wireless. In wired communication, the data (information) is communicated between two devices over wires. These wires are called a data bus. The channels (pins) on a device (or component of a device) that communicate data are collectively called a communication port.

There can be two types of wired communication ports:

1. Parallel ports: data is communicated over parallel data lines.

A parallel port is useful for fast data communication of fixed-size data. For example, the data may consist of fixed-size units of 8, 16, 32, or 64-bit, etc. Additionally, a parallel port can provide high-transfer rates with 1 bit of data-unit on each wire of the parallel data bus and the parallel port.

2. Serial ports: data is communicated as a bit-stream

A serial port can transfer data of fixed-units and communicate data of variable size. Where a parallel port uses complex circuitry for data communication, a serial port simplifies data communication by reducing the number of required channels for data transfer.

2.2 DHT-11 Humidity & Temperature Sensor

DHT11 Temperature & Humidity Sensor features a temperature & humidity sensor complex with a calibrated digital signal output. By using the exclusive digital-signal-acquisition technique and temperature & humidity sensing technology, it ensures high reliability and excellent long-term stability. This sensor includes a resistive-type humidity measurement component and an NTC temperature measurement component, and connects to a high-performance 8-bit microcontroller, offering excellent quality, fast response, anti-interference ability and cost-effectiveness. The calibration coefficients are stored as programmes in the OTP memory, which are used by the sensor's internal signal detecting process. Its small size, low power consumption and up-to-20 meter signal transmission making it the best choice for various applications, including those most demanding ones. The component is 3-pin single row pin package. (Electronics, 2022)

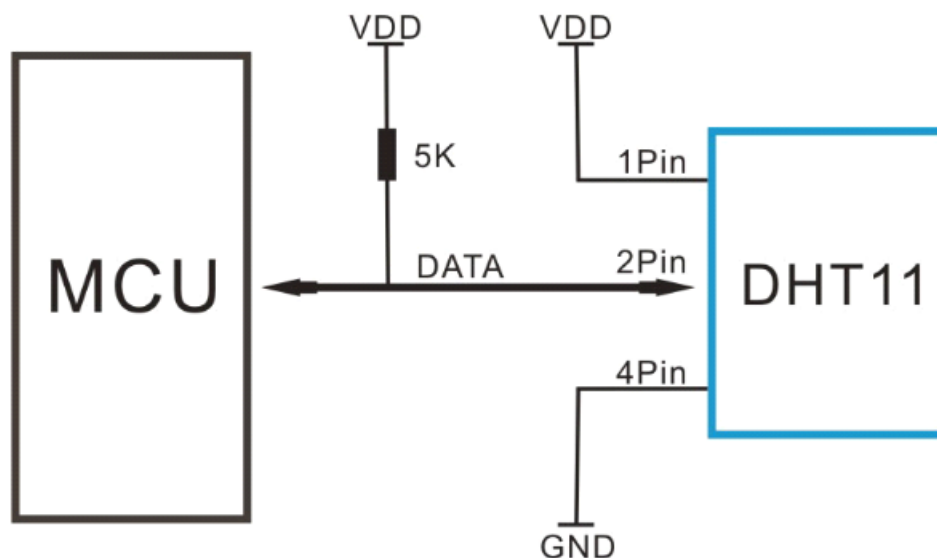


Figure 2- 3: Figure of typical application for MCU with DHT-11 sensor Connection (Electronics, 2022)

According to figure 2-3, when the connection cable is shorter than 20 meters, a 5 k Ω pull-up resistor is recommended before use. The DHT-11's power supply is 3.5 V to 5.5 V DC current. The DHT11 sensor consists of a capacitive humidity sensing element and a thermistor for sensing temperature. The humidity sensing capacitor has two electrodes with a moisture-holding substrate as a dielectric between them. Change in the capacitance value occurs with the change in humidity levels. The IC measures, processes these changed resistance values and changes them into digital form.

2.2.1 DHT-11 Sensor pin configure



Figure 2- 4: Figure of pin configuration of DHT-11 Sensor

Pin number 1 is the VCC pin and it needs to supply from 3.5 V to 5.5 V voltages for working as properly. Pin number 2 is the data pin and it gives the output as digital. DHT11 sends a response signal of 40-bit data that include the relative humidity and temperature information to ESP32. Pin number 3 is the ground pin. In the blue box with thin holes, the middle of it has a capacitive humidity sensor and a thermistor to measure the surrounding air. (Electronics, 2022)

2.3 5V Relay Module

A relay module is an electronic switching device that allows low-power microcontrollers or microprocessors to control higher-power circuits or devices. It is widely used in various electronic projects and automation systems due to its ability to isolate and protect sensitive electronic components from high voltages and currents. The relay module operates on a 5-volt power supply. Relays are electromechanical devices consisting of an electromagnet and one or more sets of contacts. When a small current flows through the coil (usually controlled by a microcontroller's GPIO pin), it generates a magnetic field that pulls the switch contacts together, completing or interrupting the circuit on the higher-voltage side. This allows the relay to act as a virtual switch, enabling control over devices such as motors, lights, appliances, and other electrical loads. (geya, 2022)

2.3.1 5V Relay module pin configuration



Figure 2- 5: Figure of pinout of 5V relay module (Barik, 2023)

- **5V Input:** The relay module operates on a 5V power supply, which is common and readily available, making it compatible with a wide range of microcontrollers and development boards.
- **Relay Switch:** The module contains one or more relays, usually indicated by the number of channels (e.g., 1-channel, 2-channel, etc.). Each channel can control a separate electrical circuit.
- **Opto-isolation:** To protect the microcontroller from potential voltage spikes or noise on the higher-voltage side, most relay modules incorporate opto-isolators. These optocouplers electrically isolate the low-voltage control circuit from the high-voltage load side.
- **LED Indicators:** Many relay modules come equipped with LEDs that provide visual feedback, indicating the status of each relay (e.g., whether it's activated or not).
- **Screw Terminals:** The module usually has screw terminals for easy and secure connections to the external devices or circuits.
- **Current Rating:** The relay module will specify the maximum current that each relay can handle. It is essential to ensure that the relay can handle the current requirements of the load you intend to control.

2.4 Transistor Switch

A transistor can be used as a solid state switch. If the transistor is operated in the saturation region then it acts as closed switch and when it is operated in the cut off region then it behaves as an open switch. The transistor operates as a Single Pole Single Throw (SPST) solid state switch. When a zero input signal applied to the base of the transistor, it acts as an open switch. If a positive signal applied at the input terminal then it acts like a closed switch. When the transistor operating as switch, in the cut off region the current through the transistor is zero and voltage across it is maximum, and in the saturation region the transistor current is maximum and voltage across is zero. Therefore, both the on – state and off – state power loss is zero in the transistor switch. (Saini, 2021)

2.4.1 Cut Off State (Open Switch)

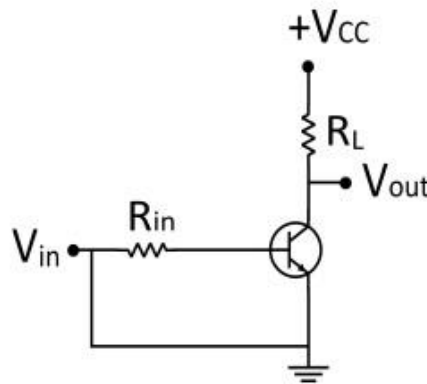


Figure 2- 6: Circuit diagram of Transistor as Open Switch mode (Saini, 2021)

When transistor operates in the cut off region shows the following characteristics –

- The input is grounded i.e. at zero potential.
- The V_{BE} is less than cut – in voltage 0.7 V.
- Both emitter – base junction and collector – base junction are reverse biased.
- The transistor is fully – off acting as open switch.
- The collector current $I_C = 0$ A and output voltage $V_{out} = V_{CC}$.

2.4.2 Saturation State (Closed Switch)

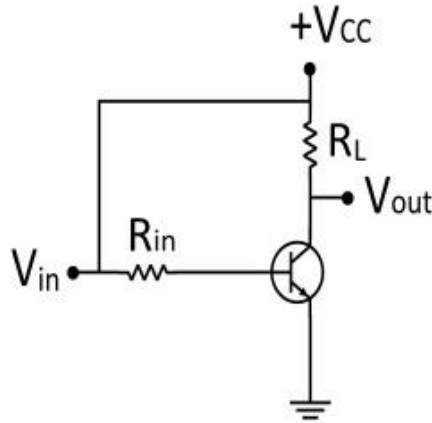


Figure 2- 7: Circuit diagram of Transistor as Close Switch mode (Saini, 2021)

The transistor operating in the saturation region exhibits following characteristics –

- The input is connected to VCC.
- Base – Emitter voltage is greater than cut – in voltage (0.7 V).
- Both the base – emitter junction and base – collector junction are forward biased.
- The transistor is fully – ON and operates as closed switch.
- The collector current is maximum
- $V_{out} = 0 \text{ V}$

$$I_c = \frac{V_{cc}}{R_L}$$

I_c = Collector current

V_{cc} = Input Voltage

R_L = Load Resistor

3 METHODOLOGY

3.1 Setting up Raspberry Pi B3

In this part, the operating system was installed into a microSD card. The ‘raspberrypi.com’ official website of Raspberry Pi was used to download the OS (Operating System) image file and ‘Raspberry Pi Imager’.

First, The web browser was opened and went to www.raspberrypi.com/software/operating-systems/ web address. Then the Raspberry Pi official download page was loaded.

Operating system images

Many operating systems are available for Raspberry Pi, including Raspberry Pi OS, our official supported operating system, and operating systems from other organisations.

Raspberry Pi Imager is the quick and easy way to install an operating system to a microSD card ready to use with your Raspberry Pi. Alternatively, choose from the operating systems below, available to download and install manually.

Download:
[Raspberry Pi OS](#)
[Raspberry Pi OS \(64-bit\)](#)
[Raspberry Pi OS \(Legacy\)](#)
[Raspberry Pi Desktop](#)

Figure 3- 1: Figure of Download page of Raspberry Pi OS and Imager

Figure 3-1 was shown the download page of Raspberry Pi OS and Imager. The OS can be downloaded by clicking each version of each OS type which has in the red box on the right-hand side of the image. The Raspberry Pi imager software can be downloaded by clicking the underlined word which has in red ellipse on the left side of the figure. Then the Raspberry Pi Imager was installed.



Figure 3- 2: Figure of Raspberry Pi Imager Software

Figure 3-2 was shown the view of Raspberry Pi Imager Software. The OS was selected after clicking the white rectangle button which has shown by numbering as 1.

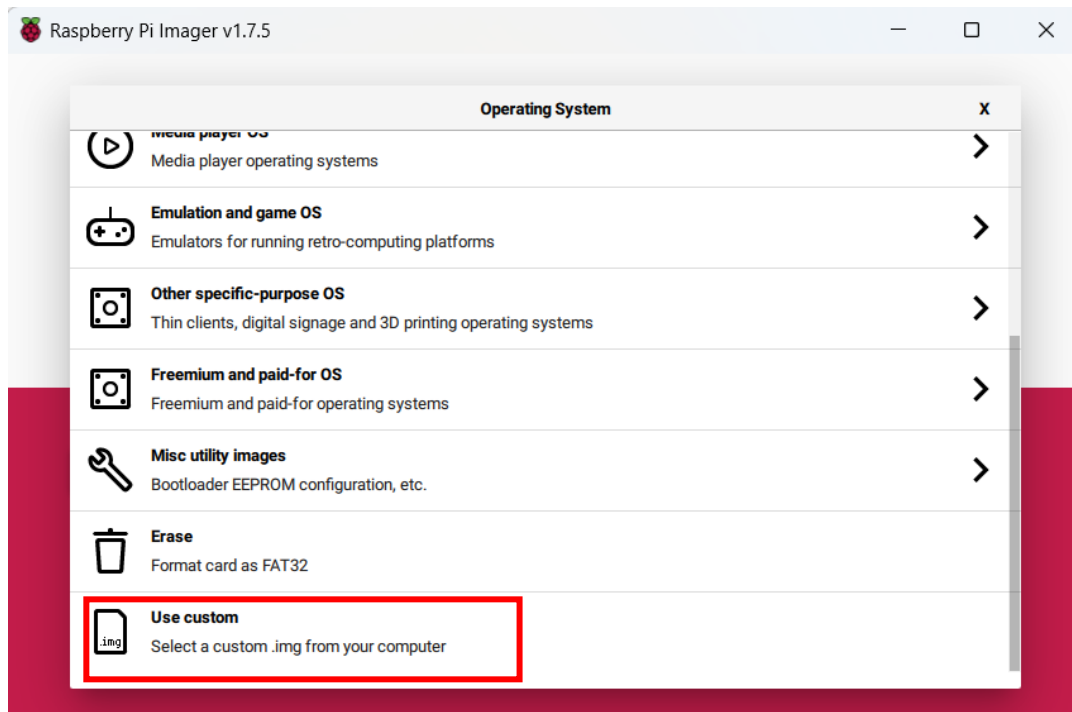


Figure 3- 3: Figure of choosing the file location of Raspberry Pi OS

The Figure 3-3 was showed the choosing the file location of Raspberry Pi OS. The OS was chosen by clicking the rectangle button which has in the red box in Figure 3-3. After selecting OS, the microSD card was plugged into the laptop and confirmed to be working properly. Then the location of the microSD card was selected by clicking the rectangle button which has shown by number 2 in Figure 3-2.

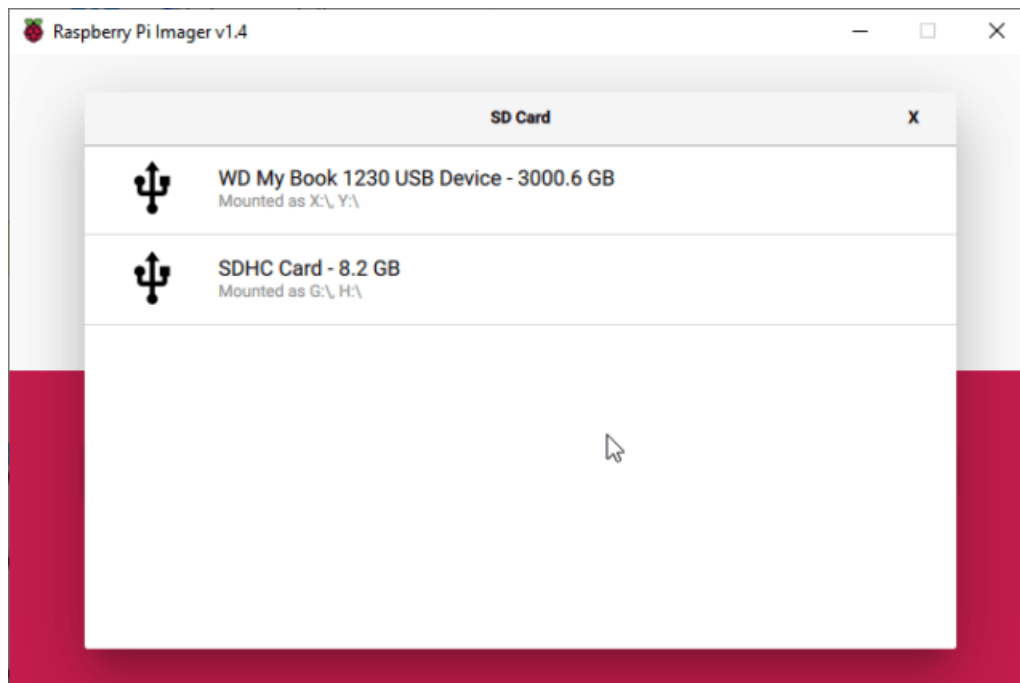


Figure 3- 4: Figure of choosing the microSD card for Installing OS

The Figure 3-4 was showed the choice of the microSD card for installing the Raspberry Pi OS.

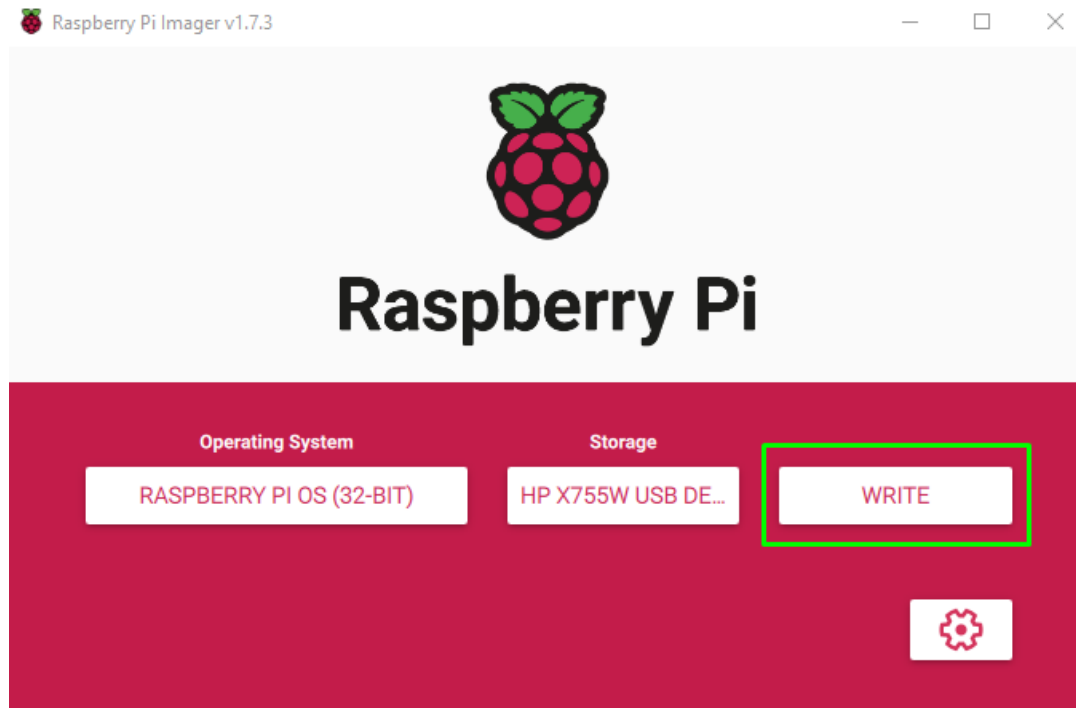


Figure 3- 5: Figure of after choosing the OS and SD card locations.

After selecting the SD card location, the rectangle button which has 'Write' shown as green colour box in Figure 3-5 was available for clicking. After clicking the write button, the OS installation was started.

Image customization options to always use

☐ Disable overscan

☐ Set hostname: raspberrypi .local

☒ Enable SSH

☒ Use password authentication

☐ Allow public-key authentication only

Set authorized_keys for 'pi':

☒ Set username and password

Username: pi

Password: ••••••••••••••••

☒ Configure wifi

SSID: Kittens

☐ Hidden SSID

Password: ••••••••••

☐ Show password

Wifi country: NZ

☒ Set locale settings

Time zone: Pacific/Auckland

Keyboard layout: us

☐ Skip first-run wizard

Persistent settings

☐ Play sound when finished

☒ Eject media when finished

☒ Enable telemetry

Figure 3- 6: Figure of adjusting the settings of OS

The Figure 3-6 was showed the settings of Raspberry Pi OS. Enable SSH radio button was selected for enabling the SSH of Raspberry Pi OS which has shown in Number 1 in Figure 3-6. The Username and Password were set by editing both text boxes which has shown in Number 2 enabling the 'Set Username and Password' radio button. The WIFI SSID and Password were added by editing both text boxes which has shown in Numbers 3 and 4 enabling the 'Configure wifi' radio button. The 'WIFI country' was selected using the given list which has shown in Number 5. The 'Time zone' and 'Keyboard layout' were added by selecting both lists which have shown in Number 6 enabling the 'Set local settings' radio button. Finally, the 'Eject media when finished' and 'Enable telemetry' were selected.

3.2 Updating Raspberry Pi OS

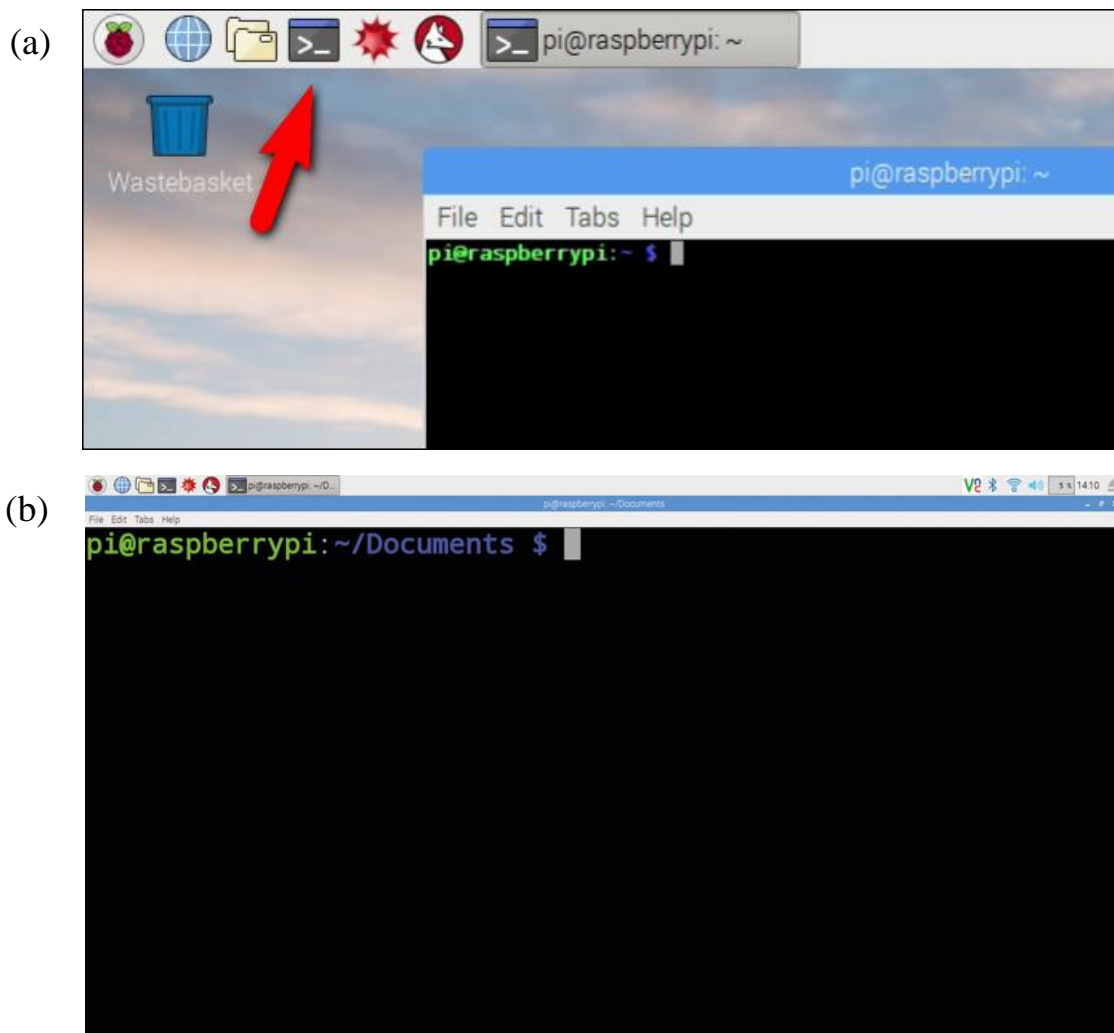


Figure 3- 7: (a) Figure of showing the Terminal of Raspberry Pi, (b) Figure of after opening the Terminal in Raspberry Pi

The mouse, keyboard, monitor and power supply were connected to Raspberry Pi B3 after plugging the microSD card. The terminal window was opened and starting update. The Terminal Icon which has in the toolbar of Raspberry Pi was shown in Figure 3-7: (a). The Terminal was shown in Figure 3-7: (b). It is mandatory to have WIFI connected.

```
sudo apt update
```

The terminal was opened and the package list was updated using above command.

```
sudo apt full-upgrade
```

The last version of packages was updated by using above command.

3.3 Blinking LED Using Python

In this part, the circuit was set up and the LED was blinked using Python code from Raspberry Pi.

3.3.1 Creating the circuit for blinking LED

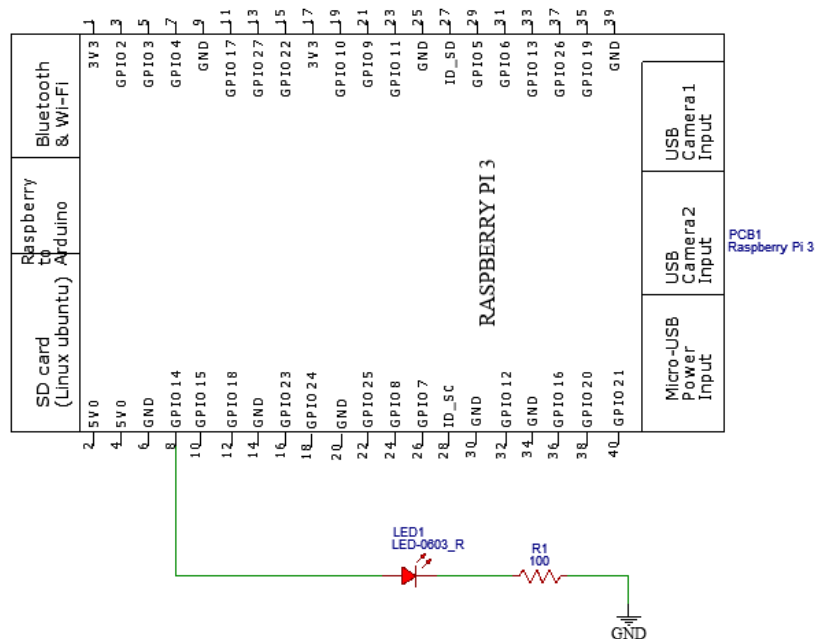


Figure 3- 8: Circuit diagram of connecting the LED with Raspberry Pi

The circuit diagram of connecting the LED with Raspberry pi was shown in Figure 3-8. The GPIO 14 (pin No 8) was connected with the LED wire. The 100 Ω resistor was connected to the LED diode and the Ground was connected to the second resistor pin.

3.3.2 Code for blinking LED from Raspberry Pi

```
1 import RPi.GPIO as GPIO
2 from time import sleep
3
4 GPIO.setwarnings(False)
5 GPIO.setmode(GPIO.BOARD)
6 GPIO.setup(8, GPIO.OUT, initial=GPIO.LOW)
7
8 while True:
9     GPIO.output(8, GPIO.HIGH)
10    sleep(1)
11    GPIO.output(8, GPIO.LOW)
12    sleep(1)
```

The *RPi.GPIO* library was imported by line number 1 and defined as *GPIO*. The *time* library was imported by line number 2 and defined as *sleep*. The GPIO warning was set up as *false* by line number 4. The GPIO set mode was set up as *GPIO.BOARD* by line number 5. The LED pin was defined as pin number 8 (without GPIO number) by line number 6 and the initial power was defined as 'low'. Then the *while* loop was defined by line number 8. The output of 8-pin power was defined as **HIGH** and **LOW** by line numbers 9 and 11. The time period was defined as **1** second between each blinking cycle.

3.4 Setting up the Circuit & getting the data from DHT-11

In this part, the circuit was created and the Temperature and Humidity values were got from the DHT-11 sensor using Python code from Raspberry Pi.

3.4.1 Creating the circuit for getting the data from DHT-11

The Raspberry pi B3, a DHT-11 humidity and Temperature sensor and wires was used for creating this circuit.

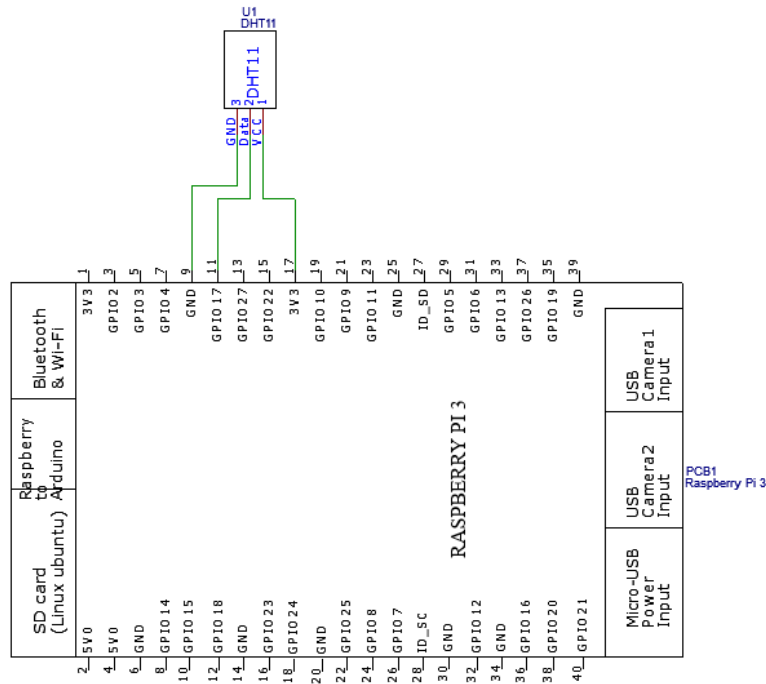


Figure 3- 9: Circuit diagram connecting the DHT-11 with Raspberry Pi

The circuit diagram of connecting the DHT-11 sensor with Raspberry Pi was shown in Figure 3-9. The GPIO 17 (pin No 11) was connected with the data pin of DHT-11. The 3.3 V power and GND pins of Raspberry Pi were connected with the VCC and GND pins of the DHT-11 sensor.

3.4.2 Import Adafruit_DHT library

The “Adafruit_DHT” library must be downloaded to use the DHT-11 sensor. The library was downloaded by using terminal which has in Raspberry Pi.

```
sudo apt install python3-pip python3-dev
```

The terminal was opened and installed required packages using above code.

```
sudo pip3 install adafruit-circuitpython
```

Then the “Adafruit library” was installed using above code.

3.4.3 Code for getting data from DHT-11 sensor

```
1 import Adafruit_DHT
2 import time
3 import datetime
4
5 #Set GPIO pin numbers
6 dht_pin = 17
7
8 #set DHT11 sensor type
9 dht_type = Adafruit_DHT.DHT11
10
11 #get current date and time
12 current_dateTime = datetime.datetime.now()
13
14 while True:
15     #Read tem & humidity data
16     humidity, temperature = Adafruit_DHT.read_retry(dht_type, dht_pin)
17
18     if humidity is not None and temperature is not None:
19         print('Temperature={0:0.1f}*C
Humidity={1:0.1f}%'.format(temperature, humidity))
20         print("Time : ", current_dateTime.strftime("%H:%M:%S"))
21         print("Date : ", current_dateTime.strftime("%Y-%m-%d"))
22         print(" ")
23
24     else:
25         print('Faield to retriev data from DHT11 sensor')
26
27
28 time.sleep(1)
```

The *Adafruit_DHT* library was imported by line number 1. The *time* library was imported by line number 2. The *datetime* library was imported by line number 3. The **DHT-11** data pin was defined as **GPIO 17** (pin number 11) by line number 6 and the DHT sensor type was defined as **DHT11** by line number 9. The current date and time were defined by line number 12. The while loop was defined by line number 14. The Humidity and Temperature data were got by **adafruit** library using pre-defined *dht_type* and *dht_pin* which has shown in line number 16. The temperature and humidity values were printed by using line number 19 with floating points number 0.1. The current date and current time were printed by using line numbers 20 and 21. Finally, the delay was set up as 1 second.

3.5 Setting up the circuit and Control Relay using DHT-11 Humidity data

In this part, the circuit was created and the Temperature and Humidity values were got from the DHT-11 sensor and controlling relay using selected humidity values using Python code from Raspberry Pi. The output voltage of GPIO is 3.3 V. The working voltage of the relay is 5 V. The switch mode of a transistor was used to control the relay.

3.5.1 Creating a circuit for getting data from DHT-11 and Controlling Relay

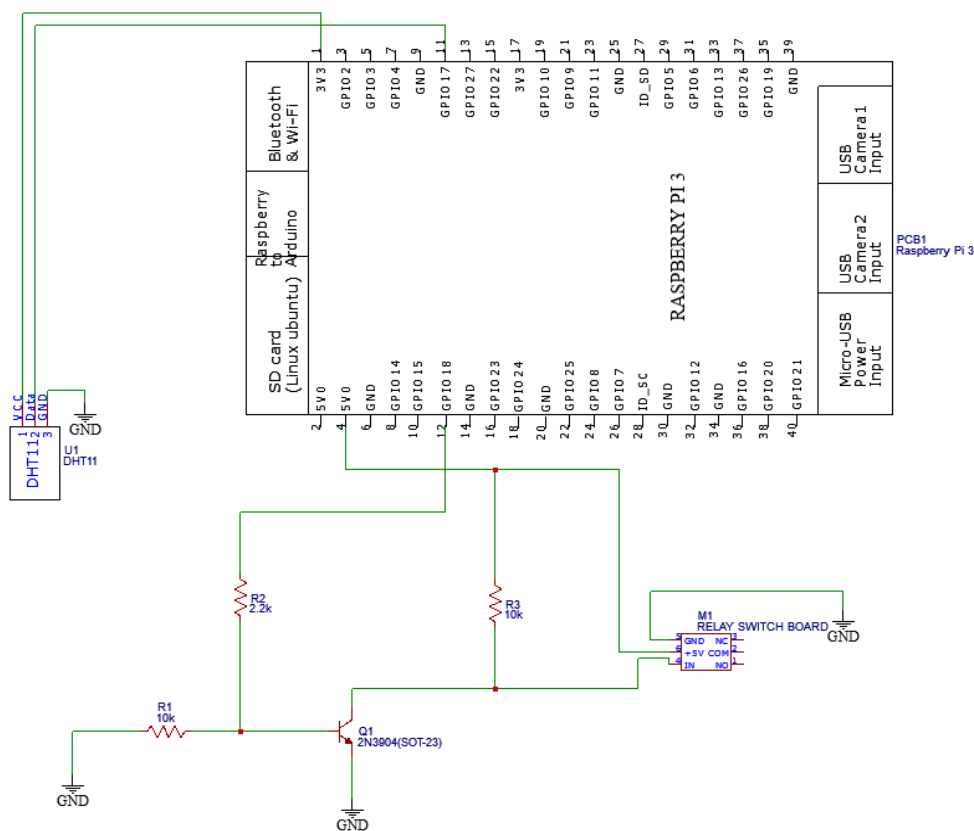


Figure 3- 10: Circuit diagram for connecting DHT-11 & Relay with Raspberry Pi

The circuit diagram of connecting the DHT-11 sensor and the relay with Raspberry Pi were shown in Figure 3-10. The GPIO 17 (pin No 11) was connected with the data pin of DHT-11. The 3.3 V power and GND pins of Raspberry Pi were connected with the VCC and GND pins of the DHT-11 sensor. The Base pin of the transistor was connected with the GPIO 18 (pin No 12) by connecting the 2.2 k Ω resistor between both the Base pin terminal and the GPIO terminal. The 5 V pin of Raspberry Pi was connected with the Collector pin of the Transistor by connecting the 10 k Ω resistor between both the Collector pin and the 5 V output pin of Raspberry Pi. The Base

pin was pulled down using a 10 k Ω resistor and the Emitter was connected to the ground pin. Then the Collector pin was connected to the control pin of the relay module. The 5 V power and GND pins of Raspberry Pi were connected with the VCC and GND pins of the Relay module.

3.5.2 Code for getting the data from DHT-11 and Controlling the Relay

```
1 import Adafruit_DHT
2 import RPi.GPIO as GPIO
3 import time
4 import datetime
5
6 #Set GPIO pin numbers
7 dht_pin = 17
8 relay_pin = 18
9
10 #set DHT11 sensor type
11 dht_type = Adafruit_DHT.DHT11
12
13 #set GPIO mode
14 GPIO.setmode(GPIO.BCM)
15
16 #set up GPIO pins
17 GPIO.setup(relay_pin, GPIO.OUT)
18
19 #get current date and time
20 current_dateTime = datetime.datetime.now()
21
22 while True:
23     #Read tem & humidity data
24     humidity, temperature = Adafruit_DHT.read_retry(dht_type, dht_pin)
25
26     if humidity is not None and temperature is not None:
27         print('Temp : {0}*C'.format(temperature))
28         print('Humi : {0}%'.format(humidity))
29     print("Time : ", current_dateTime.strftime("%H:%M:%S"))
30     print("Date : ", current_dateTime.strftime("%Y-%m-%d"))
31
32     if humidity > 70:
33         GPIO.output(relay_pin, GPIO.HIGH)
34
35     else:
36         GPIO.output(relay_pin, GPIO.LOW)
37
```

```

38         else:
39             print('Faield to retriev data from DHT11 sensor')
40
41
42         time.sleep(1)
43
44 GPIO.cleanup()

```

The *Adafruit_DHT* library was imported by line number 1. The *RPi.GPIO* library was imported by line number 2 and defined as *GPIO*. The *time* library was imported by line number 3. The *datetime* library was imported by line number 4. The **DHT-11** data pin was defined as **GPIO 17** (pin number 11) by line number 7 and the **Relay** control pin was defined as **GPIO 18** by line number 8. The DHT sensor type was defined as **DHT11** by line number 11. The GPIO set mode was set up as *GPIO.BCM* by line number 14. The Relay pin was defined as the output pin by line number 17. The current date and time were defined by line number 20. The while loop was defined by line number 22. The Humidity and Temperature data were got by **adafruit** library using pre-defined *dht_type* and *dht_pin* which has shown in line number 24. The temperature and humidity values were printed by using line number 27 & 28 with floating points number 0.1. The current date and current time were printed by using line numbers 29 and 30. Then the Relay on/off mode setting up by using humidity values. Finally, the delay was set up as 1 second and set the function for cleaning the GPIO pin.

3.6 Installing the PostgreSQL Database in Raspberry Pi

In this part, The PostgreSQL database was installed in Raspberry Pi by using terminal in Raspberry Pi. First, the Raspberry Pi was updated by using terminal command.

```

sudo apt update
sudo apt full-upgrade

```

The terminal was opened and all installed software was updated which has Raspberry Pi.

```

sudo apt install postgresql

```

Then the PostgreSQL Database was installed in Raspberry Pi using above command.

```
sudo systemctl status postgresql
```

Once the installation was completed, the PostgreSQL was automatically starting running as a service. Then the status of PostgreSQL was checked by using above command.

Then the 'psycopg2' library was installed by using command.

```
sudo apt install libpq-dev python3-dev
```

The required dependencies for 'psycopg2' are installed using above command.

```
pip install psycopg2
```

The 'psycopg' was installed using 'pip' command. If the Python 3 being used, the 'psycopg' can be installed with 'pip3' instead.

3.7 Uploading Sensor data into PostgreSQ in Raspberry PI

In this part, the temperature and Humidity data were got from DHT-11 temperature & Humidity sensor and uploaded into PostgreSQL database by using Python script. The circuit diagram was Figure 3-10.

3.7.1 Creating new database in PostgreSQL database

In this part, the new database was added into the PostgreSQL database by using Terminal Commands.

```
sudo -u postgres psql
```

The terminal was opened and the accessed to the shell using above command. Then the username and password were added.

```
CREATE DATABASE sensordata;
```

The new database called 'sensordata' database was created using above command.

```
\c sensordata;
```

The 'sensordata' database was used by using above command.

```
CREATE TABLE sensor_data(  
    current_time TIME,  
    current_date DATE,  
    temperature FLOAT,  
    humidity FLOAT  
);
```

The new data table called 'sensor_readings' was created using above command. The 'float data type was used for temperature and humidity data. The 'date' and 'time' types were used for saving the date and time data.

3.7.2 Code for uploading the sensor data into database

All of the temperature, the humidity and the relay code were same as “**3.5.2 code for getting the data from DHT-11 and Controlling the Relay**” topic. Under this topic, the data uploading to database was explained.

```
1 import Adafruit_DHT  
2 from datetime import datetime  
3 import RPi.GPIO as GPIO  
4 import psycpg2  
5 import time  
6  
7 conn = psycpg2.connect(  
8     host="localhost",  
9     database="sensordata",  
10    user="postgres",  
11    password="123"  
12 )  
13 print("starting...")  
14 GPIO.setwarnings(False)  
15 GPIO.setmode(GPIO.BCM)  
16 GPIO.setup(17, GPIO.OUT)  
17  
18
```

```

18
19 dhtPin = 4
20
21 while True:
22     try:
23         sensor = Adafruit_DHT.DHT11
24         humidity, temperature = Adafruit_DHT.read_retry(sensor, dhtPin)
25
26         #get current date and time
27         current_time = datetime.now().time().strftime('%H:%M:%S')
28         current_date = datetime.now().date()
29
30         cursor = conn.cursor()
31         cursor.execute("INSERT INTO sensor_readings (time, date, temperature,
humidity) VALUES (%s, %s, %s, %s)",
32             (current_time,current_date,temperature,humidity))
33
34         print('Temp={0:0.1f}*C Humidity={1:0.1f}%'.format(temperature, humidity))
35         print("Time : ", current_time)
36         print("Date : ", current_date)
37
38         if humidity > 71:
39             print('Relay On')
40             GPIO.output(17, GPIO.HIGH)
41
42         else:
43             print('Relay Off')
44             GPIO.output(17, GPIO.LOW)
45
46         print("")
47         conn.commit()
48         cursor.close()
49
50         time.sleep(1)
51
52     except KeyboardInterrupt:
53         break
54
55 conn.close()
56
57
58

```


The “psycpg2” library was imported by line number 4. The connection of PostgreSQL data base was defined by line number 7 to 11. The Host was defined as “localhost”. The database name, user name and password of database were defined by line number 9, 10 and 11. Then the ‘try’ function was added including to the ‘while’ loop for handling the errors and throw the exceptions. The ‘cursor’ of connection was defined by line number 30 and the database array was defined by line number 31. Data was committed by line number 47 and cursor was closed by line number 48. Then the exceptions were thrown by line number 52 as “KeyboardInterrupt” and the connection was closed by line number 55.

3.8 Installing the PostgreSQ in PC

The PostgreSQL was downloaded by using <https://www.postgresql.org/download/> link.

Downloads

PostgreSQL Downloads

PostgreSQL is available for download as ready-to-use packages or installers for various platforms, as well as a source code archive if you want to build it yourself.

Packages and Installers

Select your operating system family:



Figure 3- 11: Download page of PostgreSQL

The figure 3-11 was showed download page of PostgreSQL. Then the OS type was selected and the related web page was gone. Then the version was selected and the download file was started to download.

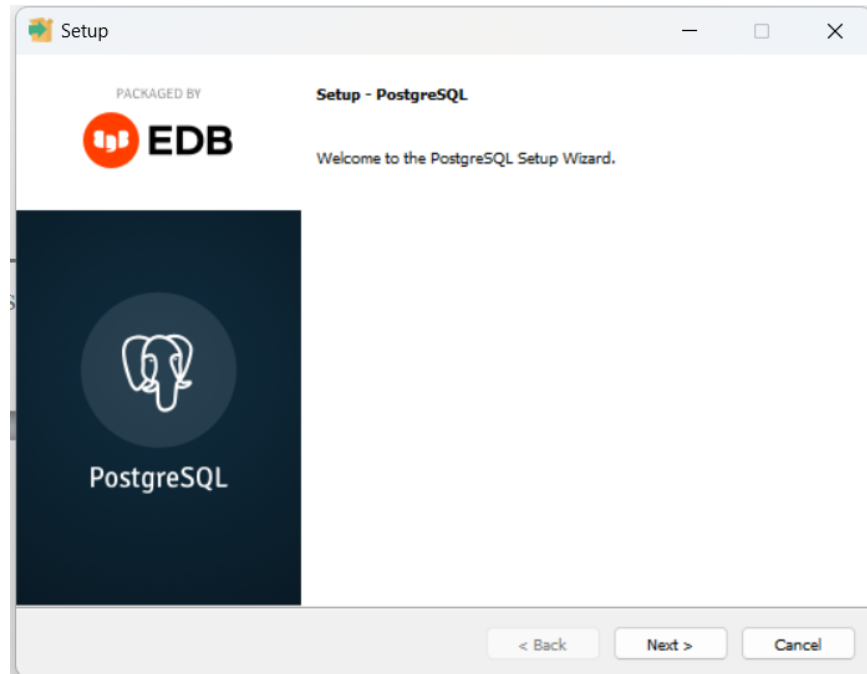


Figure 3- 12: figure of first installing step of PostgreSQL

The figure 3-12 was showed the first installing stage of PostgreSQL. The next button was clicked and goes to ahead.

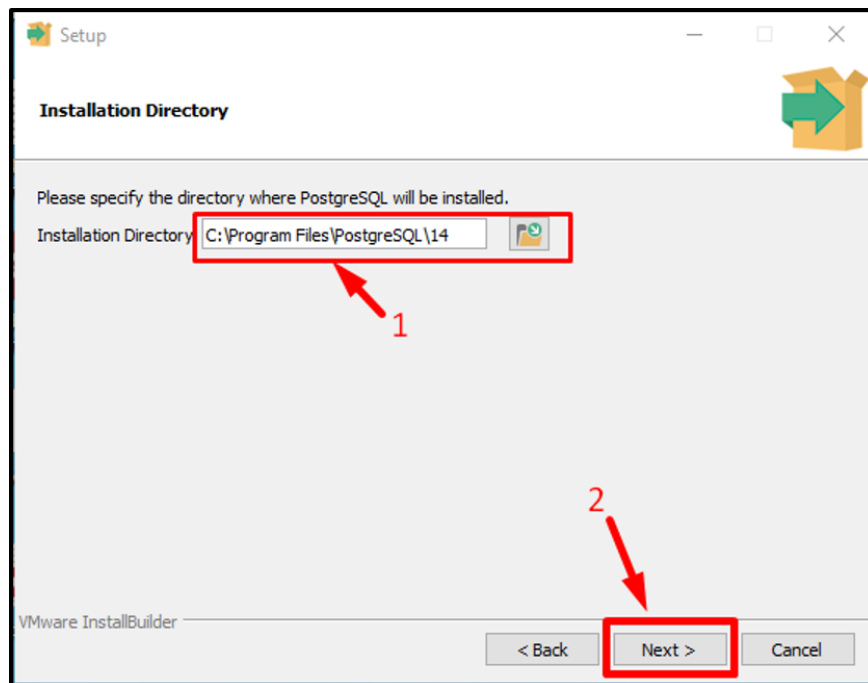


Figure 3- 13: figure of second installing step of PostgreSQL

The figure 3-13 was showed the second installing step of PostgreSQL. The path was selected for installing the PostgreSQL on a Windows PC and which has shown in number 1. Then the 'next' button was clicked which has shown in number 2.

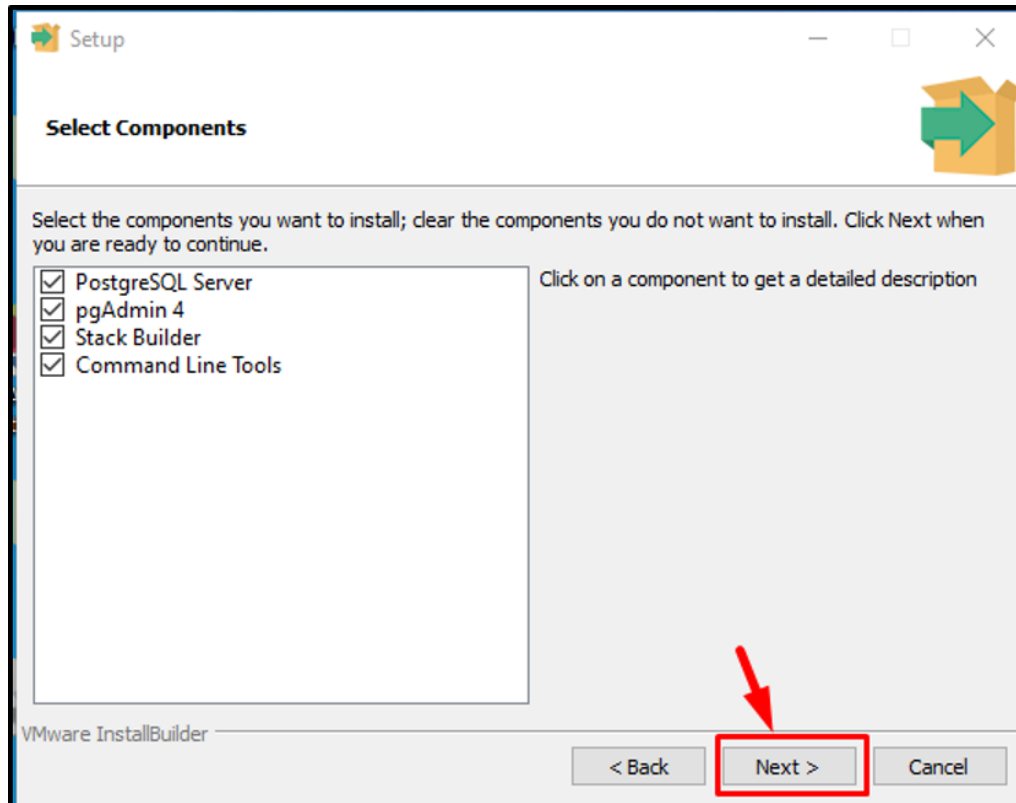


Figure 3- 14: figure of third installing step of PostgreSQL

The figure 3-14 was showed the third installing step of PostgreSQL. All four components were selected and clicked to 'next' button.



Figure 3- 15: figure of fourth installing step of PostgreSQL

The figure 3-15 was showed the fourth installing step of PostgreSQL. The data directory was selected and clicked the 'next' button.

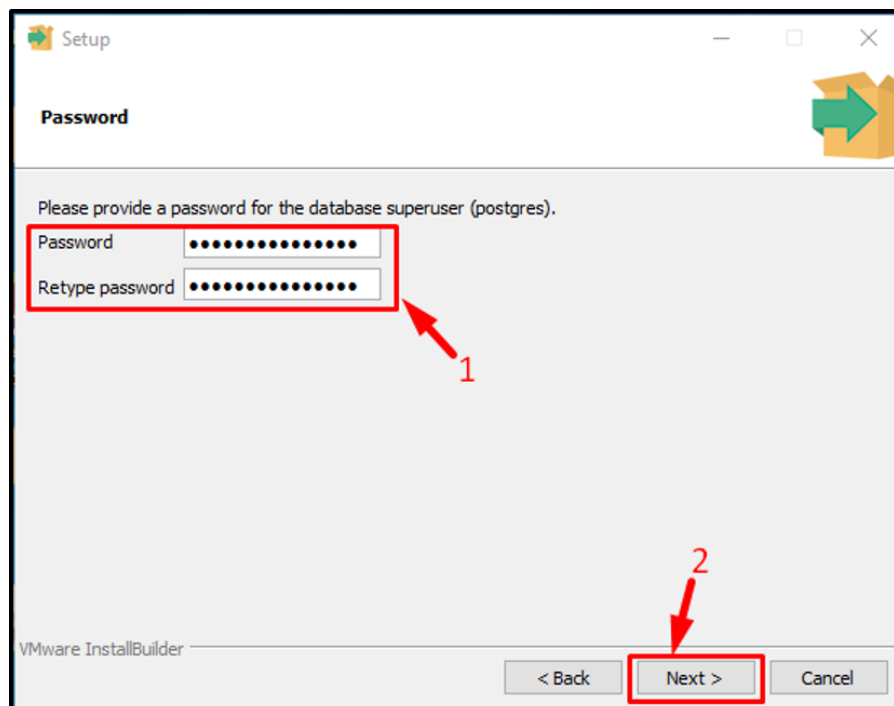


Figure 3- 16: figure of fifth installing step of PostgreSQL

The figure 3-15 was showed the fifth installing step of PostgreSQL. The username and password were added into the given spaces which have shown in number 1. The 'next' button was clicked which has shown in number 2.

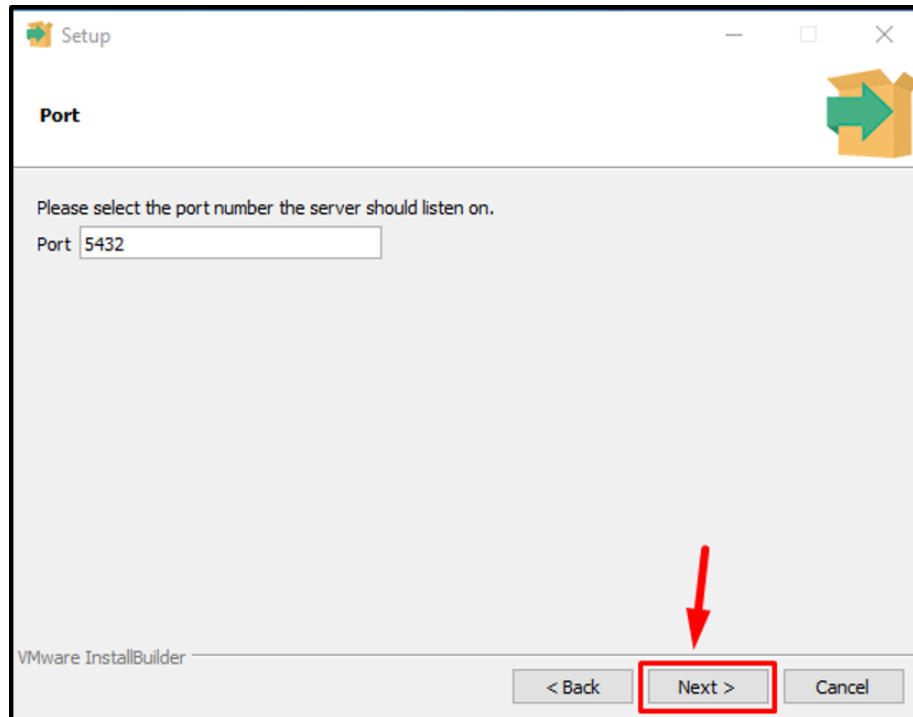


Figure 3- 17: figure of sixth installing step of PostgreSQL

The figure 3-15 was showed the sixth installing step of PostgreSQL. The 'port number' was added to the given space and the number '5432' was set as the port number. The 'next' button was clicked to going to next step.

Then the 'next' button of the all of future steps was clicked and installed the PostgreSQL. Finally, the 'finish' button was clicked.

After installing the PostgreSQL, the 'PgAdmin' was opened by searching the 'pgAdmin' in the search button of the start menu.

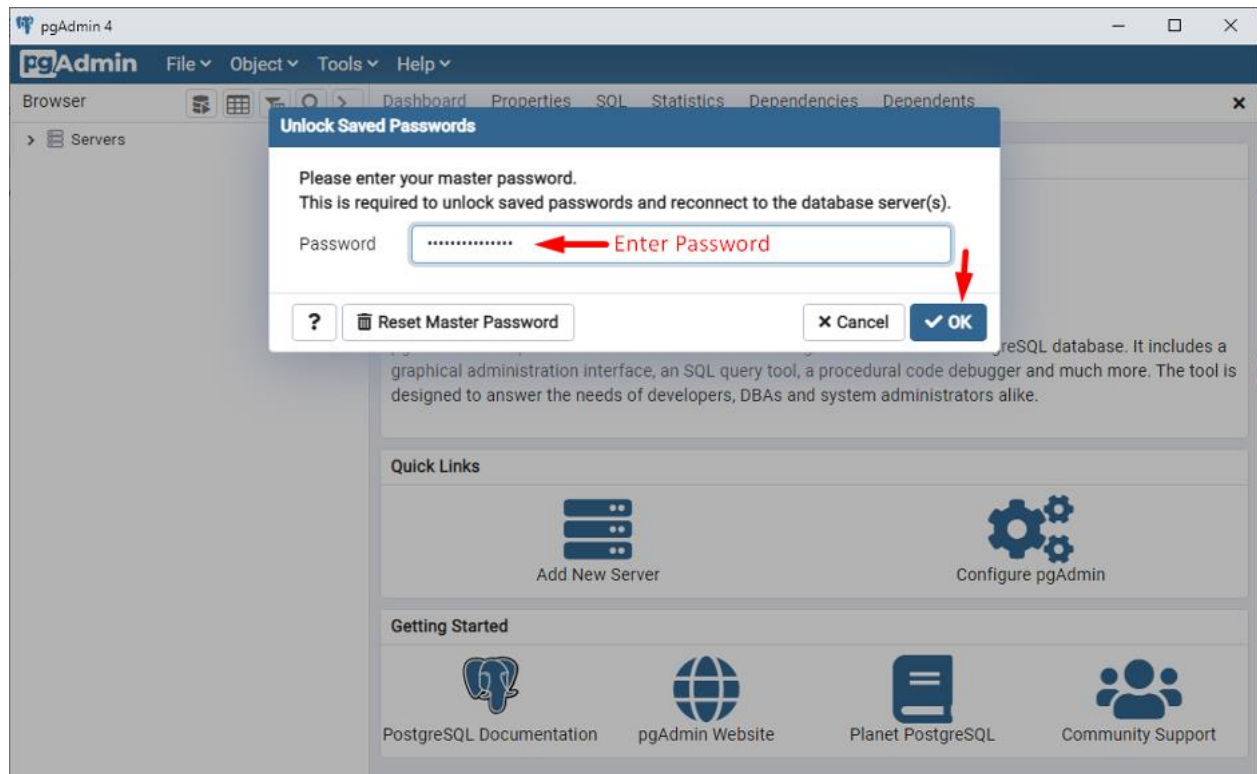


Figure 3- 18: Figure of unlocking the PostgreSQL in PgAdmin

The figure 3-18 was showed the unlocking of PostgreSQL database from PgAdmin software. The password was entered as correctly in the given space of PgAdmin and then clicked the 'Ok' button after adding the correct password.

3.9 Uploading Sensor data into PostgreSQ in PC

In this part, the temperature and Humidity data were got from DHT-11 temperature & Humidity sensor and uploaded into PostgreSQL database which has in PC by using Python script and IP address. The PC and Raspberry Pi should connect the same WIFI network for success this step. The circuit diagram was Figure 3-10.

3.9.1 Creating new database in PostgreSQL database

In this part, the new database was added into the PostgreSQL database by using PgAdmin.

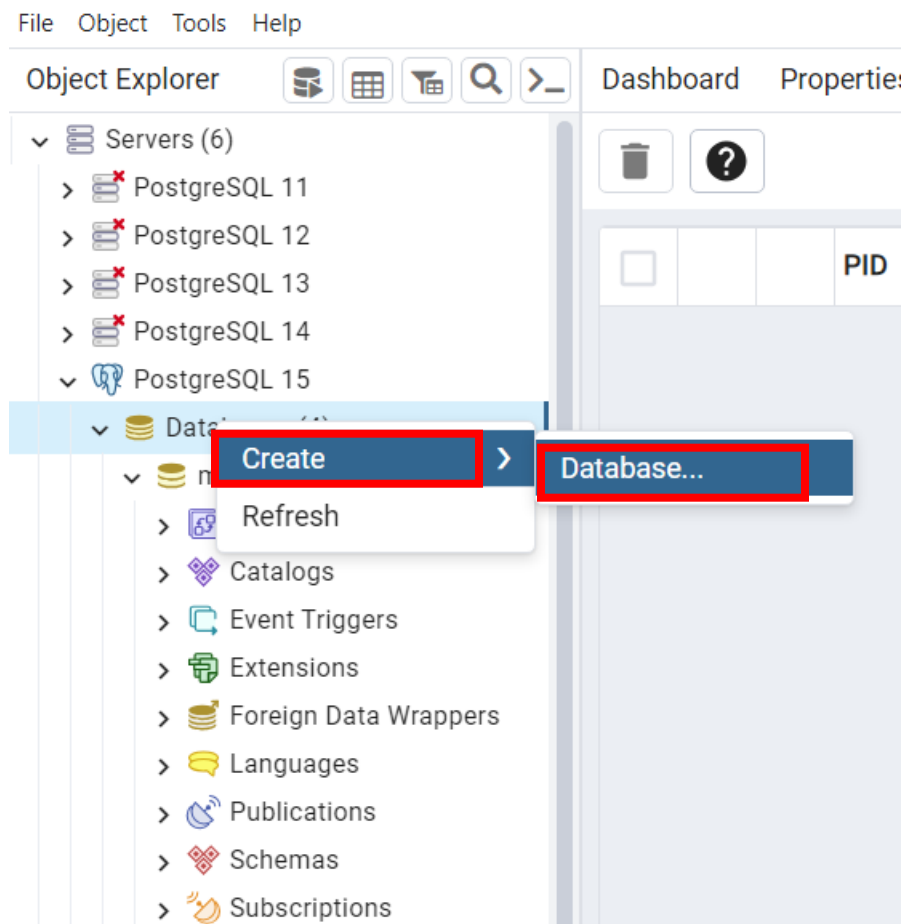
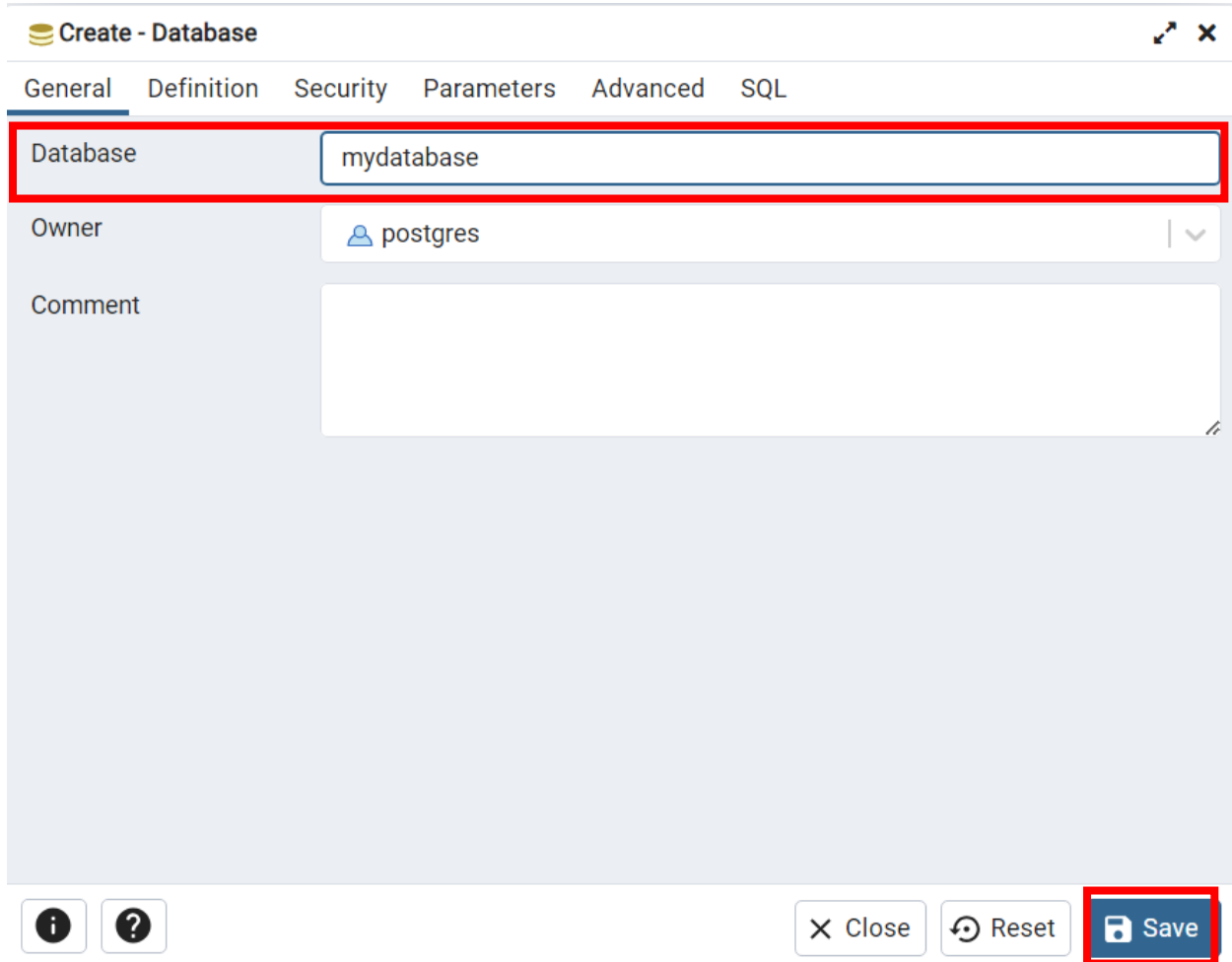


Figure 3- 19: Figure of creating new database in PostgreSQL

The figure 3-19 was shown the creating the new database of PostgreSQL using PgAdmin. First, the PgAdmin was started to run. Then the new database was created using right click the top of 'Database' and 'Create → Database'. Then the new dialogue box was opened.



The screenshot shows the 'Create - Database' dialog box in PgAdmin. The 'Database' field is highlighted with a red box and contains the text 'mydatabase'. The 'Owner' field shows 'postgres' with a dropdown arrow. The 'Comment' field is empty. At the bottom, the 'Save' button is highlighted with a red box, along with 'Close' and 'Reset' buttons.

Figure 3- 20: Figure of adding new name for database in PostgreSQL

The figure 3-20 was shown the adding new database name for creating the new database in PostgreSQL in PgAdmin. The new database name was added into given space of 'Database' word. Then the 'Save' button was clicked to save database.

```
CREATE TABLE sensor_data(  
    current_time TIME,  
    current_date DATE,  
    temperature FLOAT,  
    humidity FLOAT  
);
```

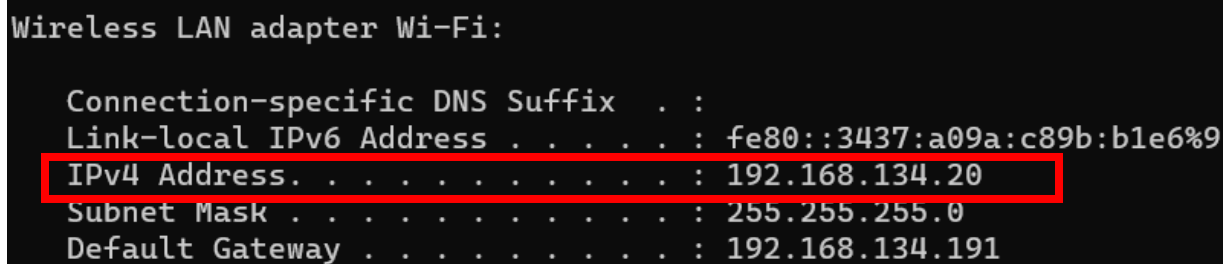

After creating a new database, the 'Query tool' was opened in PgAdmin. The above query was added to create the new table for saving the sensor data. The new data table called 'sensor_readings' was created using above command. The 'float data type was used for temperature and humidity data. The 'date' and 'time' types were used for saving the date and time data.

3.9.2 Find the IP address in Local network.

The command prompt was opened on a Windows PC using 'cmd' search keyword. Then the below command was added to the command prompt.

```
ipconfig
```

Then the IPv4 address was copied and added to the address of the host in the Python script of Raspberry Pi.



```
Wireless LAN adapter Wi-Fi:

Connection-specific DNS Suffix  . : 
Link-local IPv6 Address . . . . . : fe80::3437:a09a:c89b:b1e6%9
IPv4 Address. . . . . : 192.168.134.20
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 192.168.134.191
```

Figure 3- 21: Figure of showing the IP address in Command Prompt

The figure 3-21 was shown the all IP address of the wireless LAN adapter connected local WIFI network. The IPv4 address was used as the host IP address in Raspberry Pi.

3.9.3 Code for uploading the sensor data into database

All of the temperature, the humidity and the relay code were same as “3.5.2 code for getting the data from DHT-11 and Controlling the Relay” topic. Under this topic, the data uploading to database was explained.

```
1 import Adafruit_DHT
2 from datetime import datetime
3 import RPi.GPIO as GPIO
4 import psycpg2
5 import time
6
7 conn = psycpg2.connect(
8     host="localhost",
9     database="sensordata",
10    user="postgres",
11    password="123"
12 )
13 print("starting...")
14
15 GPIO.setwarnings(False)
16 GPIO.setmode(GPIO.BCM)
17 GPIO.setup(17, GPIO.OUT)
18
19 dhtPin = 4
20
21 while True:
22     try:
23         sensor = Adafruit_DHT.DHT11
24         humidity, temperature = Adafruit_DHT.read_retry(sensor, dhtPin)
25
26         #get current date and time
27         current_time = datetime.now().time().strftime('%H:%M:%S')
28         current_date = datetime.now().date()
29
30         cursor = conn.cursor()
31         cursor.execute("INSERT INTO sensor_readings (time, date, temperature,
32 humidity) VALUES (%s, %s, %s, %s)",
33             (current_time,current_date,temperature,humidity))
34
35         print('Temp={0:0.1f}*C Humidity={1:0.1f}%'.format(temperature, humidity))
36         print("Time : ", current_time)
37         print("Date : ", current_date)
```

```

38         if humidity > 71:
39             print('Relay On')
40             GPIO.output(17, GPIO.HIGH)
41
42         else:
43             print('Relay Off')
44             GPIO.output(17, GPIO.LOW)
45
46         print("")
47         conn.commit()
48         cursor.close()
49
50         time.sleep(1)
51
52     except KeyboardInterrupt:
53         break
54
55 conn.close()
56
57
58

```

The “psycopg2” library was imported by line number 4. The connection of PostgreSQL data base was defined by line number 7 to 11. The Host was defined as IP address which has found from PC. The database name, user name and password of database were defined by line number 9, 10 and 11. Then the ‘try’ function was added including to the ‘while’ loop for handling the errors and throw the exceptions. The ‘cursor’ of connection was defined by line number 30 and the database array was defined by line number 31. Data was committed by line number 47 and cursor was closed by line number 48. Then the exceptions were thrown by line number 52 as “KeyboardInterrupt” and the connection was closed by line number 55.

3.10 Installing the Mosquitto MQTT server in Raspberry Pi

In this part, the ‘mosquitto’ MQTT server added to the raspberry pi by using Terminal Commands.

```
sudo apt install mosquitto mosquitto-clients
```

The terminal was opened and the ‘mosquitto’ MQTT client was installed using above command.

```
sudo systemctl start mosquitto
```

The mosquito server was started by using above command.

3.11 Uploading the sensor data into database using MQTT service

In this part, the temperature and Humidity data were got from DHT-11 temperature & Humidity sensor and uploaded into PostgreSQL database which has in PC by using Python script and 'mosquitto' MQTT service. The circuit diagram was Figure 3-10.

3.11.1 Publishing data from Raspberry Pi to Mqtt Client

```
1 import Adafruit_DHT
2 from datetime import datetime
3 import RPi.GPIO as GPIO
4 import time
5 import paho.mqtt.client as mqtt
6 import json
7
8 MQTT_BROKER = "test.mosquitto.org"
9 MQTT_PORT = 1883
10
11 def on_connect(client, userdata, flag, rc):
12     print("Connected")
13
14 def on_publish(client, userdata, mid):
15     print("published")
16
17 #mqtt client setup
18 client = mqtt.Client()
19 client.on_connect = on_connect
20 client.on_publish = on_publish
21
22 print("Connection result: ", client.connect(MQTT_BROKER, MQTT_PORT, 60))
23
24 client.loop_start()
25
```

```

26 GPIO.setwarnings(False)
27 GPIO.setmode(GPIO.BCM)
28 GPIO.setup(17, GPIO.OUT)
29
30 dhtPin = 4
31 while True:
32
33     sensor = Adafruit_DHT.DHT11
34     humidity, temperature = Adafruit_DHT.read_retry(sensor, dhtPin)
35
36     #get current date and time
37     current_time = datetime.now().time().strftime('%H:%M:%S')
38     current_date = datetime.now().date()
39
40     print('Temp={0:0.1f}*C Humidity={1:0.1f}%'.format(temperature,
humidity))
41     print("Time : ", current_time)
42     print("Date : ", current_date)
43
44     if humidity >= 70:
45         print('Relay On')
46         GPIO.output(17, GPIO.HIGH)
47
48     else:
49         print('Relay Off')
50         GPIO.output(17, GPIO.LOW)
51
52     payload = {
53         "temperature" : temperature,
54         "humidity" : humidity,
55         "date_time" : time.strftime("%Y-%m-%d %H:%M:%S")
56     }
57     print(payload)
58
59     client.publish("payload", json.dumps(payload))
60     #print("")
61
62     time.sleep(10)
63

```

This part is the sending of the data from Raspberry Pi to Mosquitto Broker. The “paho.mqtt.client” library was imported as mqtt by line number 5. The ‘json’ library was imported by line number 6. The MQTT Broker address was defined as ‘test.mosquitto.org’ by

line number 8. The port number of the MQTT broker was defined as '1883' by line number 9. The broker connecting function was defined as 'on_connect' by line number 11 and the data public to broker function was defined as 'on_publish' by line number 14. Then the 'client' was defined and added client type as 'mqtt.Client()' by line number 18. Then the connection and publish functions were set up for the client by line numbers 19 and 20. Then the connection result was printed and the client loop was started. Then the data was added into the payload which got from 'json' libraries by line number 52. Then the payload data was printed by line number 57. Finally, the payload data was published with the name 'payload' by line number 59.

3.11.2 Receiving and Uploading data from Mqtt client to Database

```
1 import paho.mqtt.client as mqtt
2 import psycopg2
3 import json
4
5 MQTT_BROKER = "test.mosquitto.org"
6 MQTT_PORT = 1883
7
8 conn = psycopg2.connect(
9     host="localhost",
10    port="5432",
11    database="mqttvalues",
12    user="postgres",
13    password="123"
14 )
15
16 def on_connect(client, userdata, flags, rc):
17     client.subscribe("payload")
18     print(f"Connected")
19
20 def on_message(client, userdata, msg):
21     # Callback function for MQTT message received
22     if msg.topic == "payload":
23         payload = json.loads(msg.payload)
24         print(f'Updating database', payload)
25
26         # Create a cursor object to execute SQL queries
27         cursor = conn.cursor()
28
29         # Execute the SQL query
30         cursor.execute("INSERT INTO sensor_data (temperature, humidity,
31             date_time) VALUES (%s, %s, %s)",
32             (payload["temperature"], payload["humidity"], payload["date_time"]))
33
34 ..
```

```

32         # Commit the changes & close the cursor and database connection
33         conn.commit()
34         cursor.close()
35
36 # MQTT client setup
37 client = mqtt.Client()
38 client.on_connect = on_connect
39 client.on_message = on_message
40
41 print("Connection result:", client.connect(MQTT_BROKER, MQTT_PORT, 60))
42
43 client.loop_start()
44
45 while True:
46     pass
47

```

This part is the receiving the data from Mosquitto Broker and uploading to database. The “paho.mqtt.client” library was imported as mqtt by line number 1. The “psycopg2” library was imported by line number 2. The ‘json’ library was imported by line number 3. The MQTT Broker address was defined as ‘test.mosquitto.org’ by line number 5. The port number of the MQTT broker was defined as ‘1883’ by line number 6. The connection of PostgreSQL data base was defined by line number 8 to 12. The Host was defined as “localhost”. The database name, user name and password of database were defined by line number 10, 11 and 12. The broker connecting function was defined as ‘on_connect’ by line number 16 and the data receive message to PC function was defined as ‘on_message by line number 20. The message topic called ‘payload’ was sent from Raspberry Pi. Therefore the message topic was it. The message topic was defined by line number 22 to 24. The ‘cursor’ of connection was defined by line number 27 and the database array was defined by line number 30. Data was committed by line number 33 and cursor was closed by line number 34. Then the ‘client’ was defined and added client type as ‘mqtt.Client()’ by line number 37. Then the connection and message functions were set up for the client by line numbers 38 and 39. Then the connection result was printed and the client loop was started. Then the data was received into the payload which got from ‘json’ libraries by line number 52. Then the payload data was printed by line number 41.

4 RESULTS AND ANALYSIS

4.1 Raspberry Pi Desktop view

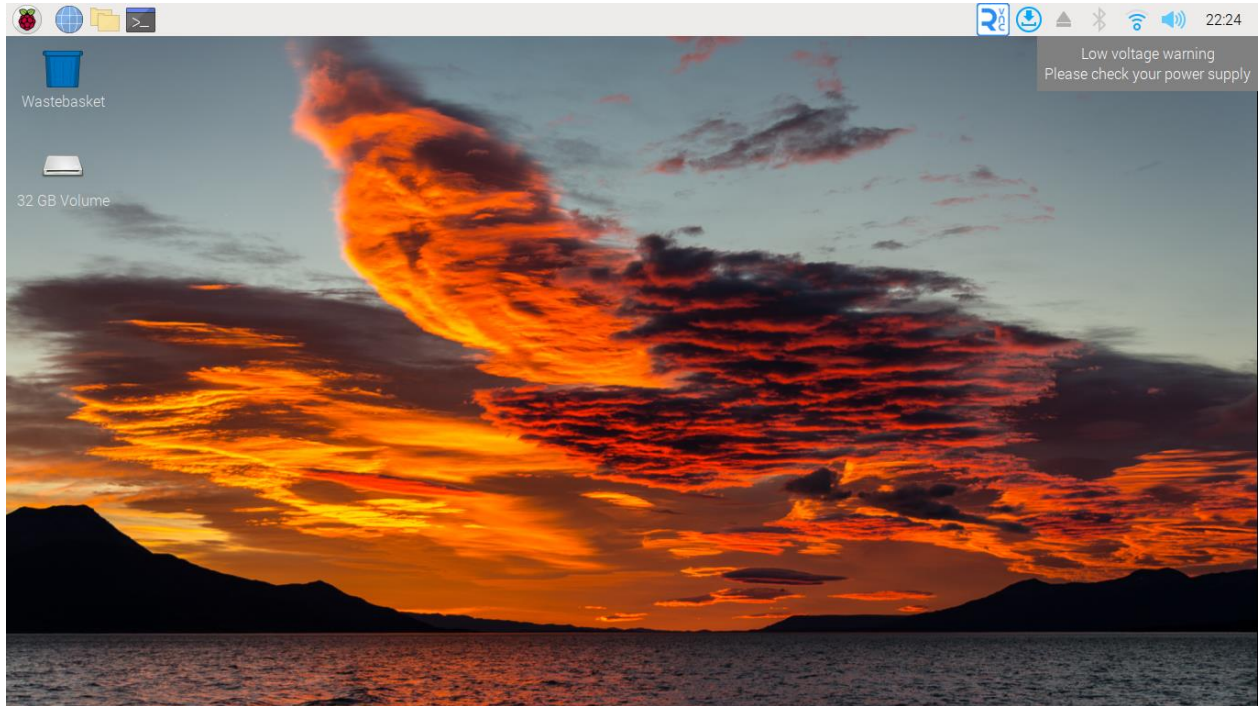


Figure 4-1: Figure of Raspberry Pi Desktop View

The figure 4-1 was shown the desktop view of Raspberry Pi B3. The main menu, Chrome, File Manager, and Terminal were located on the top left side of the taskbar. The System tray, Bluetooth, WiFi network, Sound and time were located at the top of the right on the taskbar. The wastebasket is located at the top of the left corner of the desktop.

4.2 The view of LED blinking

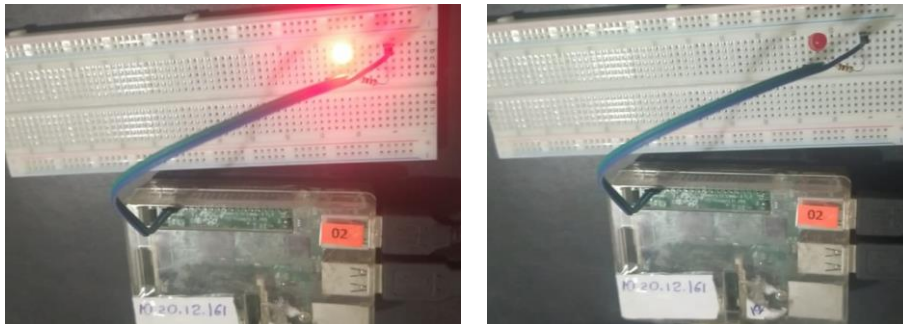


Figure 4-2: (a) Figure of ON state of LED, (b) Figure of OFF state of LED

The Figure 4-2 : (a) was shown the ON State of the LED controlled by the Raspberry Pi. Figure 4-2 : (b) was shown the OFF state of LED controlling by the Raspberry Pi. The resistor was connected with the LED pin full down, to provide the suitable voltage for LED. The time between every ON and OFF state was 1 second.

4.3 Getting data from DHT-11 and Printing in the Terminal

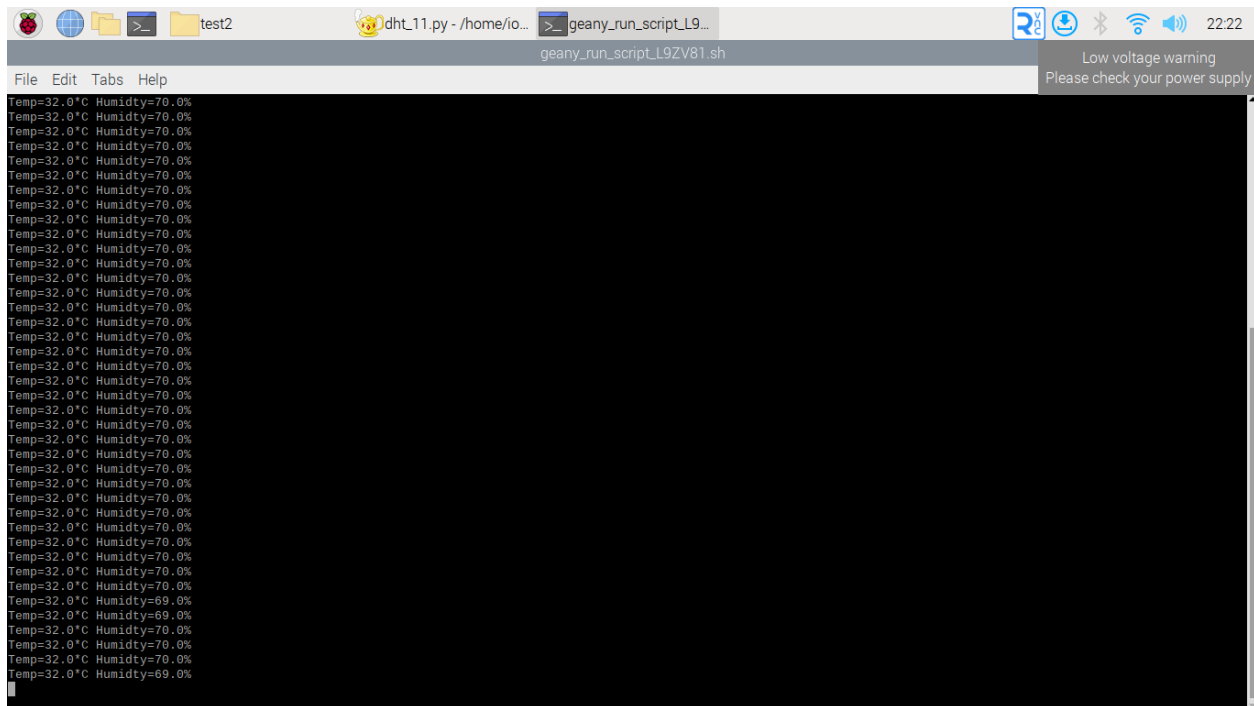
A screenshot of a Raspberry Pi terminal window. The window title bar shows the Raspberry Pi logo, a file explorer icon, and the path 'test2'. The terminal content displays a series of sensor readings: 'Temp=32.0°C Humidity=70.0%' repeated 20 times, followed by 'Temp=32.0°C Humidity=69.0%' repeated 4 times. The terminal window has a menu bar with 'File', 'Edit', 'Tabs', and 'Help'. The system tray at the bottom right shows icons for network, Bluetooth, and power, along with the time '22:22' and a 'Low voltage warning' message: 'Please check your power supply'.

Figure 4-3: Figure of getting data from DHT-11 and printing data

The figure 4-3 was shown the printing data in the terminal of Raspberry Pi. The humidity and Temperature data were printed as percentage and Celsius values.

4.4 Uploading the data into local database in Raspberry Pi

The resulting views for uploading the data into PostgreSQL local data base of the Raspberry Pi.

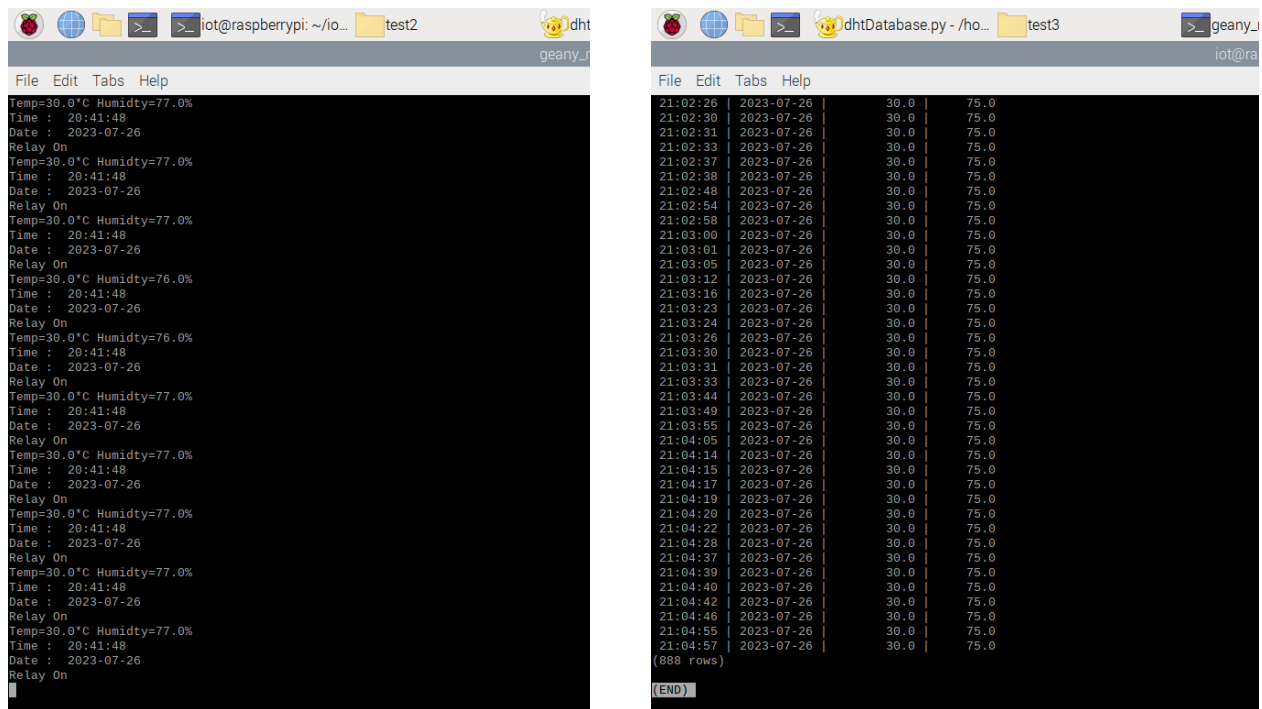


Figure 4-4:(a) Figure of printing the sensor data in terminal, (b) Figure of the local database view in Raspberry Pi

The figure 4-4 : (a) was shown the temperature, humidity, current time, current data and relay state printing in the terminal. The figure 4-4: (b) was shown the local database terminal view while uploading data.

4.5 Uploading the data into PC database using IP address

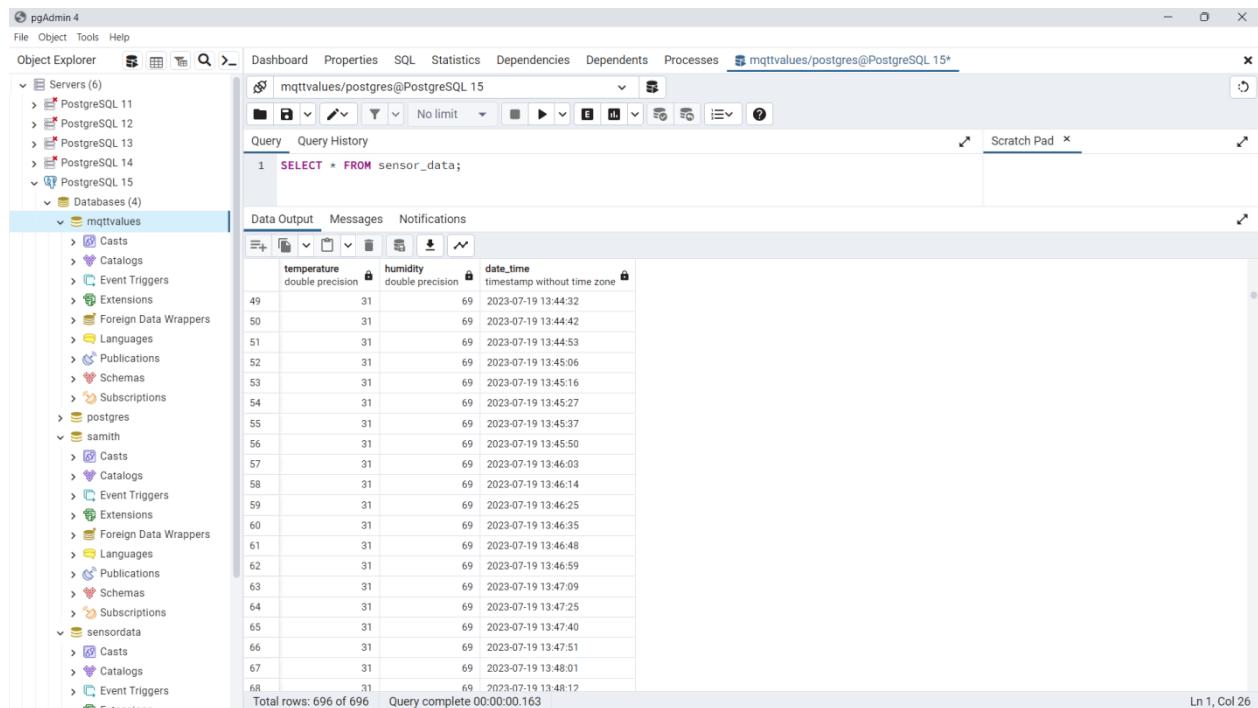
The resulting views for uploading the data into PostgreSQL PC data base from Raspberry Pi to PC. The resulted IPv4 address was '192.168.132.20'.

```
Wireless LAN adapter Wi-Fi:

    Connection-specific DNS Suffix  . : 
    Link-local IPv6 Address . . . . . : fe80::3437:a09a:c89b:b1e6%9
    IPv4 Address. . . . . : 192.168.134.20
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.134.191
```

Figure 4-5: Figure of resulting view of IP addresses

The figure 4-5 was shown the resultant view of the IP address in the command terminal of the PC. The four IP addresses were shown in the command terminal. The IPv4 address was used to transfer data from the Raspberry Pi to PC.



	temperature double precision	humidity double precision	date_time timestamp without time zone
49	31	69	2023-07-19 13:44:32
50	31	69	2023-07-19 13:44:42
51	31	69	2023-07-19 13:44:53
52	31	69	2023-07-19 13:45:06
53	31	69	2023-07-19 13:45:16
54	31	69	2023-07-19 13:45:27
55	31	69	2023-07-19 13:45:37
56	31	69	2023-07-19 13:45:50
57	31	69	2023-07-19 13:46:03
58	31	69	2023-07-19 13:46:14
59	31	69	2023-07-19 13:46:25
60	31	69	2023-07-19 13:46:35
61	31	69	2023-07-19 13:46:48
62	31	69	2023-07-19 13:46:59
63	31	69	2023-07-19 13:47:09
64	31	69	2023-07-19 13:47:25
65	31	69	2023-07-19 13:47:40
66	31	69	2023-07-19 13:47:51
67	31	69	2023-07-19 13:48:01
68	31	69	2023-07-19 13:48:12

Total rows: 696 of 696 Query complete 00:00:00.163 Ln 1, Col 26

Figure 4-6:(a) Figure of PgAdmin database data view in PC

The figure 4-6 was shown the updated data of temperature, humidity, current time and current date in PgAdmin. The three columns were there. Those are Temperature, Humidity and Current Date and Time.

4.6 Uploading the data into PC database using MQTT client

The resulting views for uploading the data into PostgreSQL PC data base from Raspberry Pi to PC. The ‘mosquitto’ MQTT broker was used for data transformations.

```

Date : 2023-07-26
Relay On
{'temperature': 30.0, 'humidity': 75.0, 'date_time': '2023-07-26 21:29:00'}
published
Temp=30.0°C Humidity=75.0%
Time : 21:29:08
Date : 2023-07-26
Relay On
{'temperature': 30.0, 'humidity': 75.0, 'date_time': '2023-07-26 21:29:08'}
published
Temp=30.0°C Humidity=75.0%
Time : 21:29:10
Date : 2023-07-26
Relay On
{'temperature': 30.0, 'humidity': 75.0, 'date_time': '2023-07-26 21:29:10'}
published
Temp=30.0°C Humidity=75.0%
Time : 21:29:13
Date : 2023-07-26
Relay On
{'temperature': 30.0, 'humidity': 75.0, 'date_time': '2023-07-26 21:29:13'}
published
Temp=30.0°C Humidity=75.0%
Time : 21:29:15
Date : 2023-07-26
Relay On
{'temperature': 30.0, 'humidity': 75.0, 'date_time': '2023-07-26 21:29:15'}
published
Temp=30.0°C Humidity=75.0%
Time : 21:29:20
Date : 2023-07-26
Relay On
{'temperature': 30.0, 'humidity': 75.0, 'date_time': '2023-07-26 21:29:20'}
published
Temp=30.0°C Humidity=75.0%
Time : 21:29:25
Date : 2023-07-26
Relay On
{'temperature': 30.0, 'humidity': 75.0, 'date_time': '2023-07-26 21:29:25'}
published

```

Figure 4-7: Figure of uploading data from Raspberry Pi to MQTT client

The figure 4-7 was shown the uploading data of temperature, humidity, current time and current date to MQTT client.

(a) Figure of local database in PC

```

mqttvalues=# SELECT * FROM sensor_data;
temperature | humidity | date_time
-----
31 | 5 | 2023-07-19 13:34:52
31 | 69 | 2023-07-19 13:35:03
31 | 69 | 2023-07-19 13:35:13
31 | 69 | 2023-07-19 13:35:24
31 | 69 | 2023-07-19 13:35:37
31 | 69 | 2023-07-19 13:35:48
31 | 69 | 2023-07-19 13:35:58
31 | 69 | 2023-07-19 13:36:11
31 | 69 | 2023-07-19 13:36:24
31 | 69 | 2023-07-19 13:36:37
31 | 69 | 2023-07-19 13:37:01
31 | 69 | 2023-07-19 13:37:14
31 | 69 | 2023-07-19 13:37:29
31 | 69 | 2023-07-19 13:37:42
31 | 69 | 2023-07-19 13:37:53
31 | 69 | 2023-07-19 13:38:03
31 | 69 | 2023-07-19 13:38:14
31 | 69 | 2023-07-19 13:38:24
31 | 69 | 2023-07-19 13:38:37
31 | 69 | 2023-07-19 13:38:48
31 | 69 | 2023-07-19 13:38:59
31 | 69 | 2023-07-19 13:39:09
31 | 69 | 2023-07-19 13:39:20
31 | 69 | 2023-07-19 13:39:30
31 | 69 | 2023-07-19 13:39:41
31 | 69 | 2023-07-19 13:39:59
31 | 69 | 2023-07-19 13:40:09
31 | 69 | 2023-07-19 13:40:22
31 | 69 | 2023-07-19 13:40:36
31 | 69 | 2023-07-19 13:40:51
31 | 69 | 2023-07-19 13:41:02
31 | 69 | 2023-07-19 13:41:17
31 | 69 | 2023-07-19 13:41:28
(33 rows)

mqttvalues=#

```

(b) Figure of receiving data from MQTT client

```

Payload inserted into database {'temperature': 31.0, 'humidity': 69.0, 'date_time': '2023-07-19 13:38:37'}
Payload inserted into database {'temperature': 31.0, 'humidity': 69.0, 'date_time': '2023-07-19 13:38:48'}
Payload inserted into database {'temperature': 31.0, 'humidity': 69.0, 'date_time': '2023-07-19 13:38:59'}
Payload inserted into database {'temperature': 31.0, 'humidity': 69.0, 'date_time': '2023-07-19 13:39:09'}
Payload inserted into database {'temperature': 31.0, 'humidity': 69.0, 'date_time': '2023-07-19 13:39:20'}
Payload inserted into database {'temperature': 31.0, 'humidity': 69.0, 'date_time': '2023-07-19 13:39:30'}
Payload inserted into database {'temperature': 31.0, 'humidity': 69.0, 'date_time': '2023-07-19 13:39:41'}
Payload inserted into database {'temperature': 31.0, 'humidity': 69.0, 'date_time': '2023-07-19 13:39:59'}
Payload inserted into database {'temperature': 31.0, 'humidity': 69.0, 'date_time': '2023-07-19 13:40:09'}
Payload inserted into database {'temperature': 31.0, 'humidity': 69.0, 'date_time': '2023-07-19 13:40:22'}
Payload inserted into database {'temperature': 31.0, 'humidity': 69.0, 'date_time': '2023-07-19 13:40:36'}
Payload inserted into database {'temperature': 31.0, 'humidity': 69.0, 'date_time': '2023-07-19 13:40:51'}
Payload inserted into database {'temperature': 31.0, 'humidity': 69.0, 'date_time': '2023-07-19 13:41:02'}
Payload inserted into database {'temperature': 31.0, 'humidity': 69.0, 'date_time': '2023-07-19 13:41:17'}
Payload inserted into database {'temperature': 31.0, 'humidity': 69.0, 'date_time': '2023-07-19 13:41:28'}
Payload inserted into database {'temperature': 31.0, 'humidity': 69.0, 'date_time': '2023-07-19 13:42:12'}
Payload inserted into database {'temperature': 31.0, 'humidity': 69.0, 'date_time': '2023-07-19 13:42:23'}
Payload inserted into database {'temperature': 31.0, 'humidity': 69.0, 'date_time': '2023-07-19 13:42:34'}

```

Figure 4-8: (a) Figure of local database in PC, (b) Figure of receiving data from MQTT client

The figure 4-8: (a) was shown the local database of PC terminal view while uploading data. The figure 4-8 : (b) was shown the temperature, humidity, current time, current data and relay state printing in the terminal receiving from mosquito mqtt client.

	temperature double precision	humidity double precision	date_time timestamp without time zone
49	31	69	2023-07-19 13:44:32
50	31	69	2023-07-19 13:44:42
51	31	69	2023-07-19 13:44:53
52	31	69	2023-07-19 13:45:06
53	31	69	2023-07-19 13:45:16
54	31	69	2023-07-19 13:45:27
55	31	69	2023-07-19 13:45:37
56	31	69	2023-07-19 13:45:50
57	31	69	2023-07-19 13:46:03
58	31	69	2023-07-19 13:46:14
59	31	69	2023-07-19 13:46:25
60	31	69	2023-07-19 13:46:35
61	31	69	2023-07-19 13:46:48
62	31	69	2023-07-19 13:46:59
63	31	69	2023-07-19 13:47:09
64	31	69	2023-07-19 13:47:25
65	31	69	2023-07-19 13:47:40
66	31	69	2023-07-19 13:47:51
67	31	69	2023-07-19 13:48:01
68	31	69	2023-07-19 13:48:12

Total rows: 696 of 696 Query complete 00:00:00.163 Ln 1, Col 26

Figure 4-9: Figure of PgAdmin database data view in PC

The figure 4-9 was shown the updated data of temperature, humidity, current time and current date in PgAdmin. The three columns were there. Those are Temperature, Humidity and Current Date and Time.

5 DISCUSSION

The purpose of this practical was to create a Raspberry Pi-based Internet of Things (IoT) project that involved data collection from a DHT-11 sensor (temperature and humidity), controlling a relay based on humidity levels, and transferring the data to a PC using both local database storage and MQTT protocol. The ultimate goal was to demonstrate a complete end-to-end solution for data monitoring and transfer in a home automation scenario.

The practical yielded promising results for the most part. The Raspberry Pi successfully controlled the LED based on the GPIO input and accurately read temperature and humidity data from the DHT-11 sensor. Using the humidity data, the Raspberry Pi effectively controlled the relay, showcasing a simple but effective automation application.

The data was appropriately displayed in the terminal, and both local and remote PostgreSQL databases received the data as intended. The integration of MQTT ensured smooth data transfer between the Raspberry Pi and the PC, further demonstrating the viability of using MQTT as a communication protocol for IoT applications.

However, during the practical, there were some instances of incorrect or inconsistent results. For example, in some cases, the relay control did not respond accurately to changes in humidity levels, which might be attributed to issues with the transistor setup or the calibration of the humidity sensor. Additionally, intermittent data transmission failures occurred while using MQTT, possibly due to network connectivity issues or incorrect configurations.

To improve the experiments, Calibration and Sensor Testing, Relay Control, MQTT Configuration, Error Handling, and Data Synchronization can do. The Sensor Inaccuracy, Unstable Data Transfer, Relay controlling lagging and data loss were observed as defects. Addressing these defects and implementing the proposed improvements will enhance the overall robustness and reliability of the Raspberry Pi IoT project.

6 CONCLUSION

In conclusion, the Raspberry Pi-based IoT project successfully showcased the integration of various components, including LED control, DHT-11 sensor data collection, relay control, local and remote database storage, and data transfer using MQTT. This practical demonstrated the Raspberry Pi's capabilities as an efficient and versatile platform for building IoT applications, enabling seamless data acquisition, control, and networking for home automation and monitoring scenarios.

However, it was evident that the accuracy of temperature and humidity data readings from the DHT-11 sensor played a crucial role in achieving reliable automation. Therefore, proper calibration and thorough testing of the sensor should be prioritized to ensure consistent and accurate results.

The implementation of MQTT for data transfer between the Raspberry Pi and the PC proved to be effective, highlighting its efficiency as a communication protocol for IoT applications. However, improvements in configuration and error handling are necessary to enhance data transfer reliability and minimize intermittent data transmission failures.

Furthermore, to ensure the precise and consistent control of the relay based on humidity levels, a refined circuit design and transistor configuration should be employed. Paying close attention to hardware setup can significantly improve the overall reliability of the system.

Additionally, data synchronization between local and remote databases needs careful management to avoid data loss or inconsistencies during data transfer. Incorporating robust error handling and logging mechanisms can facilitate the prompt identification and resolution of issues.

In conclusion, by addressing the identified areas for improvement and fine-tuning the experimental setup, the Raspberry Pi IoT project can be optimized for real-world applications, offering valuable insights into home automation, environmental monitoring, and data-driven decision-making. With continued refinement, this project holds the potential to contribute to the advancement of IoT technology and its practical applications.

7 REFERENCES

- aiven.io. (2022, 06 9). *an-introduction-to-postgresql*. Retrieved from <https://aiven.io:https://aiven.io/blog/an-introduction-to-postgresql>
- Awati, R. (2023, 07 25). *techtarget.com*. Retrieved from [whatIs.com:https://www.techtarget.com/whatis/definition/transistor](https://www.techtarget.com/whatis/definition/transistor)
- Barik, P. (2023, 02 24). *circuitdigest.com*. Retrieved from [circuitdigest.com:https://circuitdigest.com/microcontroller-projects/interface-single-channel-relay-module-with-arduino](https://circuitdigest.com/microcontroller-projects/interface-single-channel-relay-module-with-arduino)
- Electronics, M. (2022, 09 13). *Mouser*. Retrieved from www.mouser.com
- geya. (2022, 11 15). *5V relay Module*. Retrieved from www.geya.net:https://www.geya.net/5v-relay-module-how-it-works-and-application/
- mosquitto. (2022, 08 16). *mosquitto*. Retrieved from <https://mosquitto.org:https://mosquitto.org/blog/>
- Saini, M. K. (2021, 05 26). *Transistor as a switch*. Retrieved from [tutorialpoint.com:https://www.tutorialspoint.com/transistor-as-a-switch](https://www.tutorialspoint.com/transistor-as-a-switch)