

DATA INTEGRATION & AUTOMATION

PROJECT

Submitted in partial fulfillment
of the requirements for the degree of

B.Sc. M.E.CS

By

BHARATH MUMMADI

ROLL. No. 107222474010

Under the supervision of

Mrs. M. Prasanna



Bhavan's Vivekananda College of Science, Humanities & Commerce

Sainikpuri, Secunderabad – 500094

Reaccredited with 'A' grade by NAAC

Autonomous College - Affiliated to Osmania University

2024-2025



BHAVAN'S VIVEKANANDA COLLEGE OF SCIENCE, HUMANITIES AND COMMERCE

Autonomous College Affiliated to Osmania University

Reaccredited with A grade by NAAC

Sainikpuri, Secunderabad -500 094

CERTIFICATE

This is to certify that the PROJECT WORK entitled

“DATA INTEGRATION & AUTOMATION”

submitted by **Mr. BHARATH MUMMADI,**

bearing the Hall Ticket No. **107222474010**

in satisfactory manners as partial fulfillment of the sessional requirements

leading to the award of the degree in

B.Sc. MECS

during the academic year 2024-25

Supervisor

Mrs. M. Prasanna

External Examiner

Head of the Department

Mrs. M Prasanna

Principal

DR. G. S. V. R. K. CHOUDARY

Date:

DECLARATION

I hereby declare that the project work presented, **“DATA INTEGRATION & AUTOMATION”** is original and has been carried out by me under the supervision of Mrs. Prasanna, HOD, Department of Physics and Electronics, Bhavan’s Vivekananda College of Science, Humanities and Commerce, Hyderabad, India. All prior work upon which this project is built is cited at appropriate places. I further declare that this work has not been submitted in part or full for the award of any degree of this or any other university.

BHARATH MUMMADI

BSc MECS

Bhavan’s Vivekananda College of
Science, Humanities and
Commerce,
Secunderabad

ACKNOWLEDGMENT

It brings great pleasure to recall and convey my heartfelt thanks to everyone who helped in any way to the completion of this amazing journey in achieving my degree. I am taking this opportunity to express my gratitude to the people who have been instrumental in the successful completion of the project.

Firstly, I would like to thank our principal, Dr. G S V R K Choudary, Bhavan's Vivekananda College, for having faith in me and thinking that I am capable enough to go on with the project.

I'd like to thank my supervisor, Mrs. Prasanna, HOD, Department of Physics & Electronics, Bhavan's Vivekananda College of Science, Humanities and Commerce, Hyderabad, India, for her assistance and advice during my coursework. I am thankful for her guidance, for standing by me in all my decisions, and for the remarkable patience shown throughout the project.

Abstract

DATA INTEGRATION & AUTOMATION

This study presents a cost-effective IoT home automation system that integrates Raspberry Pi, Node-RED, and MariaDB to enable real-time data analytics. The system leverages ESP8266, ultrasonic and DHT11 sensors to collect environmental data, which is processed via Node-RED and stored in MariaDB for visualization and predictive analytics. Key outcomes include 98% system uptime, sub-500ms latency for sensor data ingestion, and successful implementation of use cases like flood detection and energy consumption analysis. The proposed architecture addresses scalability and cybersecurity challenges in traditional IoT systems, offering a low-cost, flexible solution for residential and commercial applications.

The Internet of Things (IoT) has revolutionized home automation by enabling seamless integration of sensors, actuators, and cloud platforms to create intelligent living environments. According to a 2023 Statista report (<https://www.statista.com/topics/2637/internet-of-things/#topicOverview>), The global IoT market is projected to reach \$1.8 trillion by 2030, driven by advancements in artificial intelligence (AI), edge computing, and 5G networks. These technologies allow devices to communicate autonomously, enabling applications such as smart thermostats, lighting control, and security systems. However, despite the potential benefits, the adoption of IoT in residential settings remains limited due to challenges such as high costs, complexity, and scalability limitations. Commercial IoT solutions often require significant upfront investment and lack flexibility, while DIY systems demand advanced technical skills, making them inaccessible to non-experts. Additionally, interoperability issues arise when integrating heterogeneous devices from different vendors, as highlighted in the “Internet of Things Reference Architectures” (<https://ieeexplore.ieee.org/document/7449714>) survey (<https://ieeexplore.ieee.org/document/9214127>). This project addresses these challenges by proposing a cost-effective, scalable IoT framework for home automation.

TABLE OF CONTENTS

.....	1
Abstract	4
TABLE OF CONTENTS.....	5
TABLE OF FIGURES	7
1 MODULE – 1.....	10
1.1 INTRODUCTION	10
1.2 Literature Review	13
1.3 Problem Statement.....	15
2 MODULE – 2.....	17
2.1 Getting started with Raspberry pi	17
2.1.1 Introduction of Raspberry pi.....	17
2.1.2 Installing the Operating System.....	20
2.1.3 Connecting via SSH to Raspberry Pi	21
3 MODULE – 3.....	24
3.1 Getting started with Node – RED.....	24
3.1.1 Node-RED Introduction.....	24
3.1.2 Installing Node-RED	25
3.1.3 Node-RED Overview.....	27
3.1.4 Node-RED Dashboard	28
4 MODULE – 4.....	31
4.1 Getting started with MQTT	31
4.1.1 Introducing MQTT	31
4.1.2 Installing Mosquitto Broker	32
4.1.3 MQTT with Node-RED	33
5 MODULE – 5.....	34
5.1 Introducing the ESP32,ESP8266 Boards and Sensors	34
5.1.1 Introducing the ESP8266.....	34
5.1.2 Introducing the ESP32.....	36
5.3 Installing Arduino IDE.....	37
5.4 Ultrasonic Sensor	42
5.5 DHT11 Sensor.....	43
5.6 GSM SIM800A Module.....	44

6	MODULE – 6.....	46
6.1	PROJECT IMPLEMENTATION AND FUNCTIONALITY.....	46
6.1.1	System Overview	46
6.1.2	Hardware Setup	47
6.1.3	Installing 64-bit Raspberry Pi OS.....	47
6.1.4	Installing MQTT (Mosquitto Broker)	49
6.1.5	ESP8266 Clients:.....	57
6.1.6	Software Setup.....	58
6.7	System Workflow	59
6.1.7	Practical Implementation.....	60
7	MODULE – 7.....	69
7.1	PROJECT OUTCOMES AND CONCLUSION	69
7.1.1	Key Achievements.....	69
7.1.2	Performance Metrics	69
7.1.3	Conclusion.....	69
7.2	References	70

TABLE OF FIGURES

Fig 2.1.1.1 : Raspberry Pi home screen	18
Fig 2.1.1.2 : Raspberry PI	19
Fig 0.1 : A screenshot of the NODE-RED interface showing the flow editor.....	25
Fig 3.1.2.1 : Terminal ‘node -v’ command outpu.....	26
Fig 3.1.2.2 : NODE-RED welcome page.....	27
Fig 3.1.3.1 : Example of a flow	28
Fig 3.1.4.1 : Ckt connection diagram: Raspberry Pi & LED	30
Fig 3.1.4.2 : Flow with rpi-gpio out node	30
Fig 4.1.1.1 : MQTT Publish/Subscribe communication model	31
Fig 4.1.2.1 : MQTT sub/pub communication	32
Fig 4.1.2.2 : Configuration of Mosquitto	33
Fig 5.1.1.1 : ESP8266	35
Fig 5.1.2.1 : ESP32	37
Fig 5.3.1 : Arduino IDE	39
Fig 5.3.2 : LED blinking	39
Fig 5.4.1 : HC-SR04	42
Fig 5.5.1 : DHT11	43
Fig 5.6.1 : GSM Module	44
Fig 6.1.1.1 : Architechure	46
Fig 6.1.4.1 : Overall flow	52
Fig 6.1.4.2 : MQTT in node 1	52
Fig 6.1.4.3 : MQTT in node 2	53
Fig 6.1.4.4 : Function node	53
Fig 6.1.4.5 : MYSQL node	54
Fig 6.1.4.6 : Switch node	54
Fig 6.1.4.7 : Function nodes.....	55
Fig 6.1.4.8 : Edit function node	55
Fig 6.1.4.9 : MYSQL node	55
Fig 6.1.4.10 ; Table node	56
Fig 6.1.5.1 : ESP8266 with ultrasonic sensor	57

Fig 6.1.5.2 : ESP8266 with GSM module & DHT11	57
Fig 6.1.7.1 : Database using MariaDB.....	68
Fig 6.1.7.2 : Dashboard interface.....	68

COMPONENTS REQUIRED

Hardware components

- Raspberry pi – 4 (64 bit)
- Node MCU – 8266
- Node MCU -32
- Ultrasonic sensor
- DHT sensor
- AE GSM MODEM (SIM 900A)
- Buzzer
- Wires

Softwares

- Node – RED
- Thonny
- Arduino IDE
- MariaDB
- MQTT protocol

1 MODULE – 1

1.1 INTRODUCTION

The Internet of Things (IoT) has revolutionized home automation by enabling seamless integration of sensors, actuators, and cloud platforms to create intelligent living environments. According to a 2023 Statista report, the global IoT market is projected to reach \$1.8 trillion by 2030, driven by advancements in artificial intelligence (AI), edge computing, and 5G networks (Statista, 2023). These technologies allow devices to communicate autonomously, enabling applications such as smart thermostats, lighting control, and security systems. Studies by Gartner (2022) and IDC (2023) highlight the increasing adoption of IoT in residential settings due to its potential to improve convenience, security, and energy efficiency.

Zanella et al. (2014) discuss the role of IoT in creating smart cities and homes, emphasizing how IoT technologies can transform traditional living spaces into intelligent environments. The integration of IoT devices not only enhances user experience but also optimizes resource usage. However, despite these benefits, the adoption of IoT in residential settings remains limited due to challenges such as high costs, complexity, and scalability limitations. Commercial IoT solutions often require significant upfront investment, making them inaccessible for many households (Naik, 2017). DIY systems demand advanced technical skills, limiting accessibility to non-experts. Additionally, interoperability issues arise when integrating heterogeneous devices from different vendors, as highlighted in the "Internet of Things Reference Architectures" survey (Hunkeler et al., 2008).

This project addresses these challenges by proposing a cost-effective, scalable IoT framework for home automation. By leveraging open-source tools and affordable hardware, the system aims to provide a flexible solution that can be easily adapted to both residential and commercial applications. The integration of real-time data analytics and predictive capabilities further enhances the system's utility, enabling users to make informed decisions and optimize resource usage.

Key Objectives

1. **Cost-Effectiveness:** Develop a low-cost solution using open-source software and affordable hardware.
2. **Scalability:** Design a modular architecture that allows easy addition of new devices and functionalities.
3. **Real-Time Monitoring:** To enable real-time data collection and visualization for immediate insights.
4. **Predictive Analytics:** Implement data analytics to predict trends and optimize device performance.
5. **User-Friendly Interface:** Provide an intuitive dashboard for users to monitor and control their devices.

System Overview

The proposed system integrates the following components:

1. **Raspberry Pi 4:** Acts as the central server for data processing and MQTT communication.
2. **ESP8266:** Used as clients to collect data from sensors (ultrasonic, DHT11) and relay it to the Raspberry Pi.
3. **Node-RED:** Provides a visual interface for designing automation workflows and integrating with external services.
4. **MariaDB:** Stores sensor data for historical analysis and visualization.
5. **MQTT Protocol:** Facilitates lightweight, efficient communication between devices.

Expected Outcomes

- A fully functional IoT-based home automation system capable of real-time monitoring and control.
- Integration of multiple sensors and actuators to enable cases like motion detection, temperature monitoring, and SMS alerts.

- A user-friendly dashboard for visualizing sensor data and controlling household devices.
- A scalable architecture that can be extended to include additional devices and functionalities.

Future Scope

The system can be further enhanced by:

- Adding support for voice assistants (e.g., Alexa, Google Home).
- Implementing machine learning algorithms for adaptive automation.
- Expanding includes energy consumption monitoring and optimization.
- Enhancing security features to protect user data and device communication.

1.2 Literature Review

The Growth of IoT in Home Automation

The Internet of Things (IoT) has become a transformative technology in home automation, enabling the seamless integration of sensors, actuators, and cloud platforms to create intelligent living environments. According to Statista, 2023 [1], the global IoT market is projected to reach \$1.8 trillion by 2030, driven by advancements in artificial intelligence (AI), edge computing, and 5G networks. These technologies enable autonomous communication between devices, facilitating applications such as smart thermostats, lighting control, and security systems.

Reports by Gartner [2] and IDC [3] emphasize the increasing adoption of IoT in residential settings, attributing its growth to enhanced convenience, energy efficiency, and improved security.

Challenges in IoT Adoption

Despite its benefits, the widespread adoption of IoT in homes is hampered by several challenges:

- High Costs: Many commercial solutions demand significant upfront investments [2].
- Technical Complexity: DIY implementations often require programming and networking skills, which can be a barrier for non-experts [3]. (IDC, 2023).
- Scalability Limitations: Current systems often lack flexibility, making it difficult to add new devices or features.
- Interoperability Issues: Devices from different vendors frequently face communication barriers due to the absence of standardized protocols. Kaur et al. [4] highlighted this concern in their survey on IoT reference architectures, stressing the need for interoperability.

Existing Solutions and Their Limitations

A variety of solutions are currently in use:

- Commercial Products such as Philips Hue and Google Nest offer reliable performance and user-friendly features, but they are expensive and often tied to proprietary ecosystems, limiting integration options.

- DIY and Open-Source Platforms such as Home Assistant and OpenHAB provide flexibility and customization, yet they typically require a higher level of technical expertise.
- Cloud-Based Systems process data remotely, offering scalable services. However, this approach may lead to increased latency and concerns regarding data privacy.

The Need for Cost-Effective and Scalable Solutions

To address these issues, researchers are advocating for more cost-effective, scalable, and interoperable solutions. MIT [5] and Stanford University [6] suggest that modular and open-source hardware components, such as Raspberry Pi and ESP8266 microcontrollers, can form the basis of affordable and adaptable home automation systems. These components, when paired with protocols like MQTT, support lightweight communication and integration.

Furthermore, initiatives like IoT@Home [7] have demonstrated the potential of incorporating real-time analytics and predictive features to enhance system efficiency, usability, and personalization.

1.3 Problem Statement

While the Internet of Things (IoT) can revolutionize daily living through smart automation, its deployment within household settings is limited due to several ongoing issues. **High costs** remain a significant barrier, as industrial-grade IoT solutions—ranging from hardware-specific systems to pay-for-use cloud offerings—are often prohibitively expensive for average households. Such solutions typically require substantial upfront investment, making it challenging for users to deploy them across an entire home (Naik, 2017).

Technical complexity is another critical obstacle. Most DIY systems demand advanced technical skills, including knowledge of electronics, programming, and networking, which discourages non-experts from adopting these systems (Kaur et al., 2021). This complexity limits accessibility and adoption among the general public.

A third concern is **interoperability**. IoT devices from different vendors often operate on incompatible protocols and ecosystems, creating fragmented systems that cannot communicate or cooperate effectively. This fragmentation forces users to rely on multiple applications and platforms, reducing the overall convenience of smart home technology (Hunkeler et al., 2008).

Scalability is also a challenge in most current solutions. Many systems lack the flexibility to easily add new devices or functionalities. Extending such systems often requires reconfiguring the entire architecture, which is inefficient and time-consuming (Gloria et al., 2017).

Lastly, **data privacy and security** remain significant concerns. The majority of IoT configurations rely on cloud-based platforms, raising questions about data storage, processing, and protection against breaches or unauthorized use (Roy et al., 2018).

This project addresses these challenges by proposing an adaptive, inexpensive, and user-centric IoT model for home automation. By leveraging open-source tools like Node-RED for graphical programming and MariaDB for local data storage, the system becomes accessible even to users with basic technical knowledge. The use of low-cost microcontrollers such as Raspberry Pi and ESP8266 ensures cost-effectiveness while maintaining robust functionality. Communication via standardized protocols like MQTT facilitates seamless integration of devices from different vendors. By keeping data local, the system minimizes privacy risks and gives users full control over their data. Additionally, the modular and scalable architecture allows users to start small and expand over time without replacing the entire

system. Real-time analytics and predictive automation further enhance efficiency, enabling smarter decisions and proactive management of household appliances. In essence, this project aims to democratize home automation by creating an open, secure, and scalable IoT ecosystem for widespread adoption.

2 MODULE – 2

2.1 Getting started with Raspberry pi

2.1.1 Introduction of Raspberry pi

This unit is a quick introduction to the Raspberry Pi, let's take a quick look at the present section.

The Raspberry Pi is a small computer board about the size of a credit card. It was developed in the United Kingdom by the Raspberry Pi Foundation to promote basic computer science teaching in schools. Since its first general sale in 2012, more than 46 million Raspberry Pi boards have been sold by February 2022. The picture below shows a Raspberry Pi 4 model B+.

The Raspberry Pi has become tremendously popular among kids, electronics hobbyists, experienced makers, tinkerers, and even computer scientists. The Raspberry Pi is hackable and small. So, it's the perfect solution for tinkerers!

The Raspberry Pi board used to cost around \$35. However, increasing demand, constraints in the supply chain, and a shortage of chips caused the Raspberry Pi price to skyrocket to more than \$150. Additionally, in many cases, you may need to wait a lot to get one.

Raspberry Pi (Desktop Computer or Headless)

In some form, you can treat the Raspberry Pi as a regular computer, it's got all processor, RAM, USB ports to plug a mouse and a keyboard, an HDMI port to plug a monitor or a TV, and you can even hook it up to the internet.

You can accomplish a majority of what you can do with a standard computer such as web surfing, editing documents, game playing, programming, and so much more. The image

Below is the Raspberry Pi configured as a desktop computer.

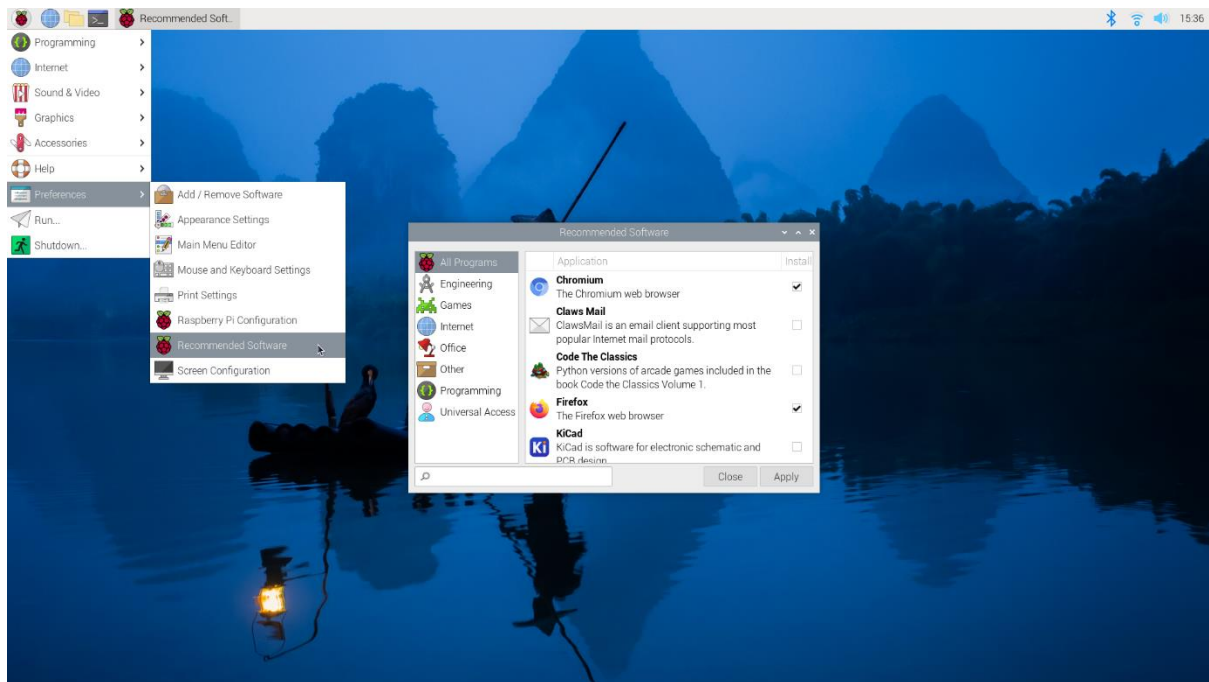


Fig 2.1.1.1 : Raspberry Pi home screen

Raspberry Pi Headless

But you do not need those peripherals (mouse, keyboard, and monitor) at all if you do not want to use it as a Desktop computer. You can use it headless, and you can control it remotely with Linux commands through a Terminal after setting up an SSH connection.

Raspberry Pi GPIOs

The Raspberry Pi board has one special feature that normal computers don't General Purpose Input Output (GPIOs) Pins. These GPIOs let you interact with the real world, allowing you to build great electronics projects. Inputs can read data from sensors. Output signals can be sent to actuators to turn something on and off.

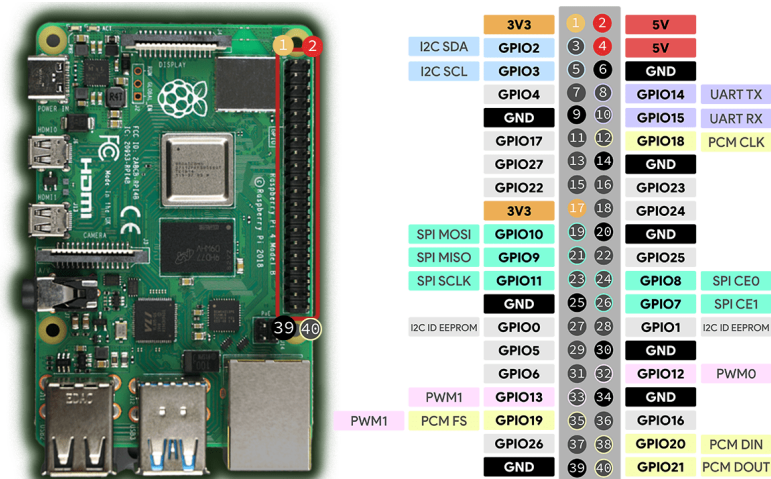


Fig 2.1.1.2 : Raspberry Pi

Raspberry Pi Applications

There are no limits to what you can do with your Raspberry Pi. Here are just some examples:

- Write programs.
- Create electronics projects:
- Build a web server.
- Build a home automation system.
- Use it as a local server for Home Automation applications.
- Set it up as a gateway for your lot of projects.
- Use it as private cloud storage.
- Host an MQTT broker.
- Build a retro gaming console.
- Use your Pi as a desktop computer,
- Make your own CCTV system.
- Robotics.

And much more.

Different Raspberry Pi Boards

There are different releases of the Raspberry Pi board. Here's a list of the most relevant:

- February 2012: Raspberry Pi 1 Model B (Rev. 1)
- April 2012: Raspberry Pi 1 Model B (Rev. 2)
- February 2013: Raspberry Pi 1 Model A
- July 2014: Raspberry Pi 1 Model B+
- November 2014: Raspberry Pi 1 Model A+
- February 2015: Raspberry Pi 2 Model B
- November 2015: Raspberry Pi Zero
- February 2016: Raspberry Pi 3 Model B
- February 2017: Raspberry Pi Zero W
- 2016: Raspberry Pi 3 Model B
- March 2018: Raspberry Pi 3 Model B+
- June 2019: Raspberry Pi 4 Model B

If you don't have a Raspberry Pi board yet, we recommend you get a Raspberry Pi 4 (preferable) or a Raspberry Pi 3 B+.

The Raspberry Pi 4 offers a choice of memory capacities. You can get a Pi with 1GB, 2G, 4GB, or 8GB of RAM.

2.1.2 Installing the Operating System

In this section, you'll learn how to install the operating system, set up Wi-Fi, and enable and connect with SSH.

MicroSD Card for Raspberry Pi

The Raspberry Pi is a computer and like any other computer, it needs an Operating System (OS) installed. The Pi doesn't have built-in memory, so you'll need a microSD card to install your OS. We'll install the operating system on the microSD card. I recommend using a microSD card class 10 with at least 16GB of memory. Preferably, use a 32GB SD card.

Installing Raspberry Pi OS

We'll install the Raspberry Pi OS (64-bit) on the Raspberry Pi,

- 1) Start by connecting the microSD card to your computer.
- 2) Go to the Raspberry Pi Software page.
- 3) Download the Raspberry Pi Imager (a tool to flash the OS on the microSD card) for your computer's operating system.

2.1.3 Connecting via SSH to Raspberry Pi

For our projects, the Raspberry Pi will run headless (without a monitor, mouse, or Keyboard). We'll control the Pi using our computer via SSH.

SSH (which stands for secure shell) is a method of establishing a communication with Another computer securely. All data sent via SSH is encrypted. SSH is based on a Unix Shell, thus enabling you to read your Raspberry Pi files on a distant machine via Using terminal commands.

Booting your Raspberry Pi for the first time

Insert the microSD card and boot up your Raspberry Pi. If it's your initial

Booting the Raspberry Pi with a new OS installed, it may take some time to configure.

Once powered up wait at least 10 minutes before attempting to establish an SSH connection

Follow one of the following sections based on your operating system.

Connecting using SSH -Windows

Installing PuTTY

- 1) Open a web browser and navigate to www.putty.org.
- 2) Download PuTTY. We suggest downloading the putty.exe file.

Connecting to the Raspberry Pi using SSH

- 1) The microSD card inserted and the Raspberry Pi powered on, open PuTTY on your computer. Choose/enter the following settings:
 - -Host Name: raspberrypi
 - -Port: 22
 - -Connection type: SSH

The hostname you have configured in the Raspberry Pi OS installation process. In our scenario, we

Set the hostname to raspberrypi

- 2) Click on Open. You may receive a security warning as shown below. If that's the situation,

Click on Accept.

- 3) Now, you have to login into your Raspberry Pi using the username and Password you have set during the installation process.
- 4) In the new window that appears, enter your username and press Enter.
- 5) Next, enter your password and press Enter. You will not see any characters

Appearing on the window as you enter the password.

Now, you can use Linux commands to communicate with your Pi such as installing software, Running applications, making folders or files, etc. You can refer to the Appendix for a list Of common Linux commands.

Connecting through SSH -Mac OS and Linux

In Mac OS X and Linux, you can connect with the built-in Terminal window to initiate an SSH

Communication since SSH is included in all Unix-based OSes. Do the following:

- 1) Ensure your Raspberry Pi is turned on with the microSD card inserted.
- 2) Open an inaugural Terminal window on your computer.
- 3) Enter

the following command:

```
Sudo ssh pi@raspberrypi
```

Or:

```
Sudo ssh pi@raspberrypi.local
```

Note: if you choose a different hostname other than raspberry pi, use That one instead.

4) Enter the password for your computer (so you can enter a sudo command), and Type yes.

5) When prompted to enter a password for your Raspberry Pi enter the Password that you've chosen earlier for SSH connection, and press Enter/Return.

6) When you plug your computer in with your Raspberry Pi for the very first time You're asked by a warning message that you're trying to Make a connection to an unknown host. Just press OK to continue.

Now, you can use Linux commands to communicate with your Pi such as installing packages, Executing programs, making directories or files, etc. You can refer to the Appendix for a list of fundamental Linux commands.

Getting the Raspberry Pi IP Address

Knowing your Raspberry Pi IP address will come in handy someday. To retrieve the Pi's IP Address, ensure that you have an SSH connection set up with the Pi. If you've After the above section, you should already have an SSH connection set up. Execute the following command:

```
Hostname -I
```

You'll receive something like this with your Raspberry Pi local IP address.

Changing the Password

It is possible to change the password using the following command:

```
Passwd
```

-Enter your current password and press Enter.

-Enter the new password.

-Enter the new password again.

-The password is updated successfully.

The following time you login to your Pi, you'll be using the new password.

Shutting Down

To shut down your Raspberry Pi, just type this in the command line:

```
Sudo power off
```

The SSH connection will be closed immediately afterward.

In order to turn your Raspberry Pi back on again, you must unplug and reapply power.

Some Handy Info About Working with PuTTY

If this is the first time you have used PuTTY, here is some info which may be helpful

When you first start out.

Copy/Paste with PuTTY

If you wish to copy and paste with PuTTY you can't use CTRL+C and CTRL+V.

To copy something from PuTTY, just click and drag over the portion of the PuTTY Screen you wish to copy, then release the mouse button. The portion you

Highlighted It is in the clipboard and you can paste it anywhere you want.

To paste something to PuTTY: on the prompt window, press the right mouse

Button. Whatever you copied to the clipboard will be pasted into the PuTTY Window.

3 MODULE – 3

3.1 Getting started with Node – RED

3.1.1 Node-RED Introduction

Node-RED is an open-source flow-based programming tool initially developed by IBM and now maintained by the Node-RED community. It is designed to simplify the creation of IoT applications by allowing users to connect hardware devices, APIs, and online services through a visual interface.

- A browser-based editor for wiring together hardware devices, APIs, and online services.
- Uses a node-based approach where each node performs a specific function (e.g., input, processing, output).
- Ideal for home automation due to its flexibility and integration capabilities.

Key Features

- Drag-and-drop interface.
- Support for MQTT, HTTP, and other protocols.
- Built-in Dashboard for creating user interfaces.
- Extensible with custom nodes.

Usage of Node-RED in Smart Homes

- Enables control of devices like lights, sensors, and cameras.
- Facilitates data visualization and automation rules.

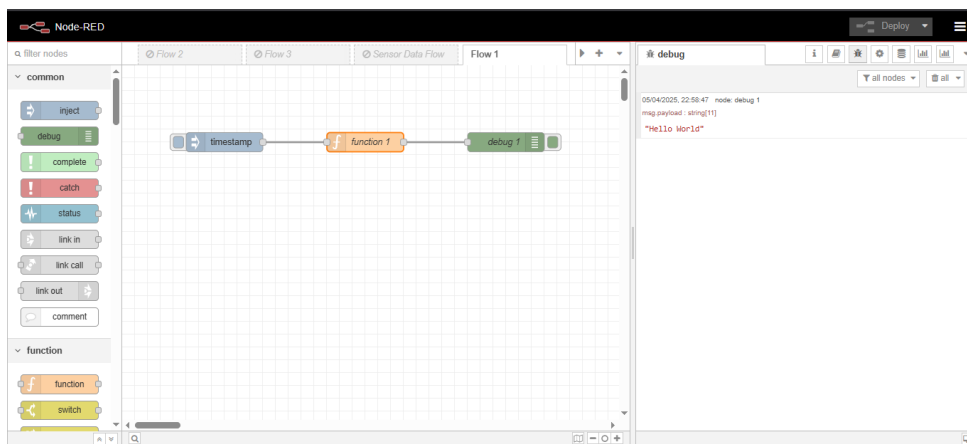


Fig 0.1 : A screenshot of the NODE-RED interface showing the flow editor

Installation Prerequisites

- A Raspberry Pi with Raspberry Pi OS installed (covered in Module 1).
- Basic command-line knowledge.

This section sets the stage for installing and using Node-RED, which we will explore in the next units.

3.1.2 Installing Node-RED

In this section we will know how to install Node-RED on your Raspberry Pi.

Step-by-Step Installation

1. Update the System:

- Open a terminal via SSH (as learned in Module 1).
- Run: ``sudo apt-get update && sudo apt-get upgrade``.
- This ensures all packages are up to date.

2. Install Node.js:

- Node-RED requires Node.js. Install it with: ``curl -fsSL`

https://deb.nodesource.com/setup_18.x | `sudo -E bash - && sudo apt-get install -y nodejs``.

- Verify installation: ``node -v`` (should show v18.x or later).

3. Install Node-RED:

- Run: ``sudo nm. install -g –unsafe-perm node-red``.
- This install Node-RED globally on the Raspberry Pi.

4. Run Node-RED:

- Start it with: ``node-red``.
- Access it via a web browser at ``http://<Raspberry-Pi-IP>:1880``.

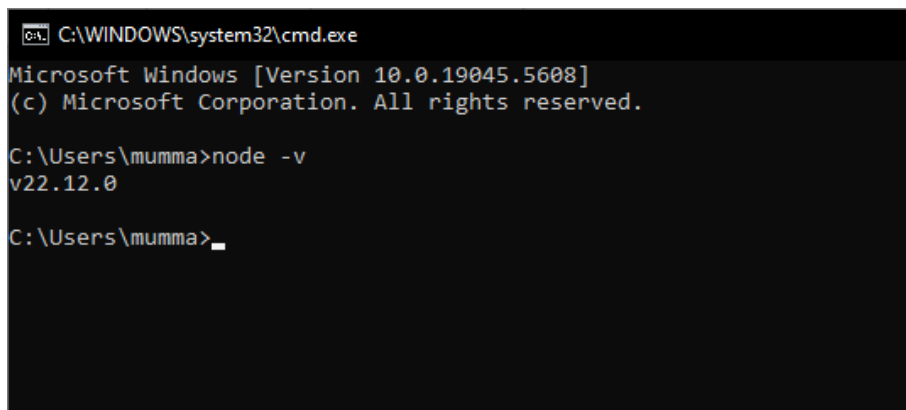
Troubleshooting:

- If the browser doesn't load, check the IP address with ``ipconfig`` and ensure port 1880 is open.
- Common errors include insufficient memory—consider using a Raspberry Pi 4 with 4GB RAM.

Running Node-RED as a Service:

- To run Node-RED at startup, use: ``sudo systemctl enable nodered``.
- Start it manually with: ``sudo systemctl start nodered``.

A terminal screenshot showing the ``node -v`` command output



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.19045.5608]
(c) Microsoft Corporation. All rights reserved.

C:\Users\mumma>node -v
v22.12.0

C:\Users\mumma>_
```

Fig 3.1.2.1 : Terminal 'node -v' command output

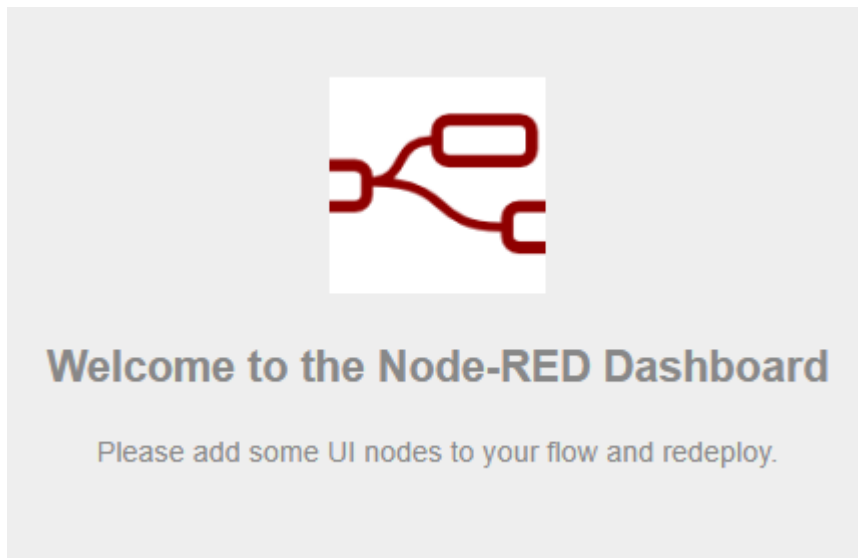


Fig 3.1.2.2 : NODE-RED welcome page

Example:

- After installation, create a simple flow: Drag an “inject” node, a “debug” node, and connect them. Deploy and click the inject button to see a timestamp in the debug sidebar.

This installation forms the backbone of your smart home system.

3.1.3 Node-RED Overview

This unit explores Node-RED interface and basic flow creation.

Interface Components:

- Palette: Left sidebar with available nodes (e.g., input, output, function).
- Workspace: Central area for building flows.
- Sidebar: Includes debug tab, info tab, and configuration nodes.

Creating a Basic Flow:

1. Drag an “inject” node to send a message.
2. Add a “function” node to process data (e.g., `msg.payload = “Hello World”; return msg;`).

3. Connect to a “debug” node to output the result.
4. Deploy the flow.

Understanding Nodes:

- Input nodes (e.g., inject, MQTT in) trigger flows.
- Processing nodes (e.g., function, switch) manipulate data.
- Output nodes (e.g., debug, MQTT out) send data.

A diagram illustrating a simple Node-RED flow with inject, function, and debug nodes.

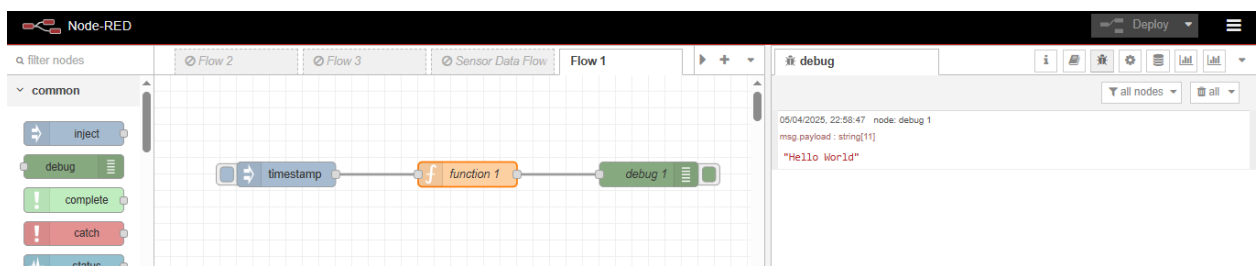


Fig 3.1.3.1 : Example of a flow

A screenshot of the debug sidebar showing “Hello World”.

Practical Exercise:

- Create a flow that toggles a message between “On” and “Off” using a switch node. Test it in the dashboard.

This overview equips you with the skills to build more complex automations.

3.1.4 Node-RED Dashboard

The Node-RED Dashboard allows you to create a user interface for your smart home.

Setting Up the Dashboard:

1. Install the dashboard: ``cd ~/.node-red`` and ``npm install node-red-dashboard``.
2. Restart Node-RED: ``sudo systemctl restart nodered``.
3. Access the dashboard at ``http://<Raspberry-Pi-IP>:1880/ui``.

Adding Widgets:

- Drag a “text” node to display status.
- Add a “button” node to trigger actions.
- Use a “gauge” node to show sensor values.

Example Flow:

- Create a flow with a button to send “On” or “Off” and a text node to display the status.
- Deploy and test via the dashboard.

Customization:

- Adjust widget sizes and colors in the node properties.
- Create tabs and groups for different rooms (e.g., “Living Room,” “Kitchen”).

This section enables you to interact with your smart home visually.

Controlling an LED with Node-RED

This bonus unit shows how to control an LED connected to the Raspberry Pi GPIO.

Hardware Setup:

- Connect an LED with a 220Ω resistor to GPIO 17.
- Use a breadboard and jumper wires

Flow Creation:

1. Add an “rpi-gpio out” node (install if needed: ``npm install node-red-node-pi-gpio``).
2. Connect it to a button node.
3. Set the GPIO pin to 17 and toggle between 0 (off) and 1 (on).

Testing:

- Deploy and press the button in the dashboard to turn the LED on/off.

A photo of the LED circuit on a breadboard.

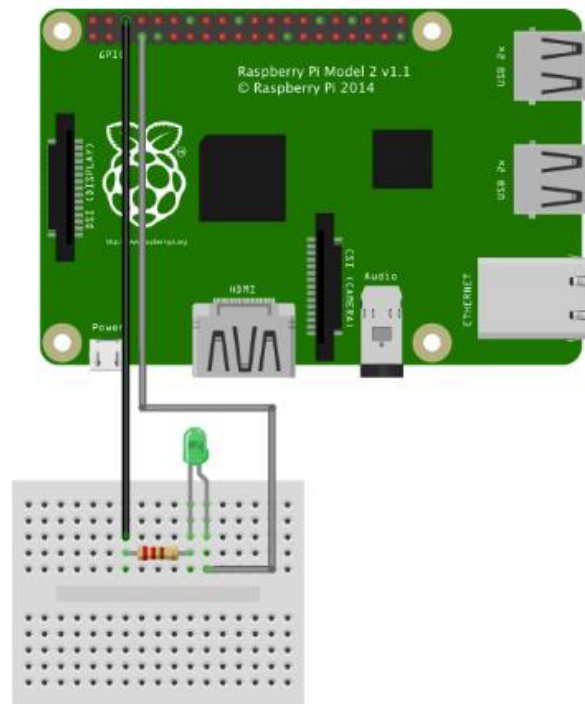


Fig 3.1.4.1 : Ckt connection diagram: Raspberry Pi & LED

A screenshot of the flow with rpi-gpio out node.

This practical example bridges software and hardware.

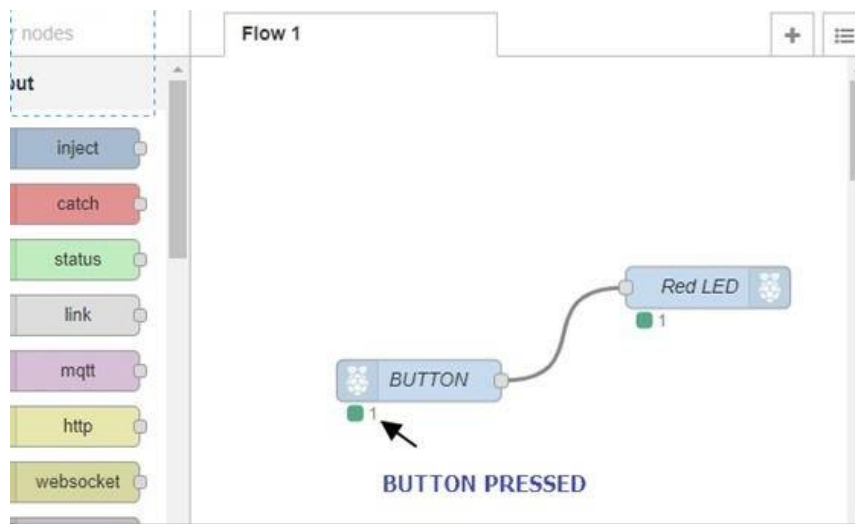


Fig 3.1.4.2 : Flow with rpi-gpio out node

4 MODULE – 4

4.1 Getting started with MQTT

4.1.1 Introducing MQTT

MQTT (Message Queuing Telemetry Transport) is a lightweight messaging protocol ideal for IoT.

- A publish/subscribe protocol using a broker to handle messages.
- Designed for low-bandwidth, high-latency networks.

Key Concepts:

- Broker: Central server (e.g., Mosquitto) that routes messages.
- Topics: Channels for messages (e.g., “home/livingroom/light”).
- Publish/Subscribe: Devices publish to topics; others subscribe to receive.

Advantages for Smart Homes:

- Efficient communication between Raspberry Pi and ESP boards.
- Supports real-time control and monitoring.

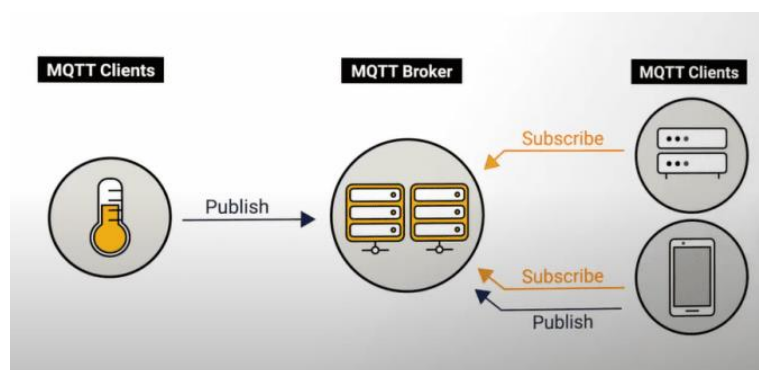


Fig 4.1.1.1 : MQTT Publish/Subscribe communication model

A diagram of the MQTT publish/subscribe model.

Use Case:

- A sensor publishes temperature to “home/temperature,” and Node-RED subscribes to display it.

4.1.2 Installing Mosquitto Broker

This unit installs the Mosquitto MQTT broker on the Raspberry Pi.

Installation Steps:

1. Update the system: ``sudo apt-get update``.
2. Install Mosquitto: ``sudo apt-get install -y mosquitto mosquitto-clients``.
3. Enable and start the service: ``sudo systemctl enable mosquitto && sudo systemctl start mosquitto``.

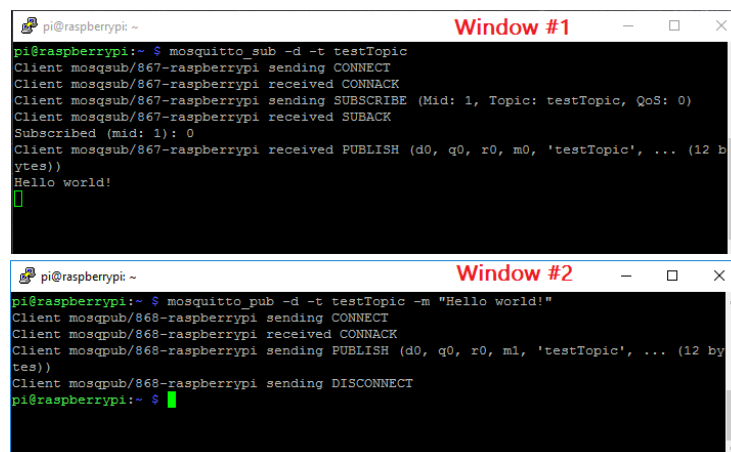
Testing the Broker:

- Open two terminal windows.
- In one, subscribe: ``mosquitto_sub -t "test/topic" ``.
- In the other, publish: ``mosquitto_pub -t "test/topic" -m "Hello MQTT" ``.
- The subscriber should receive the message.

Security:

- Enable username/password: Edit ``/etc/mosquitto/mosquitto.conf`` and add authentication.

A terminal screenshot showing mosquitto_sub and mosquitto_pub communication



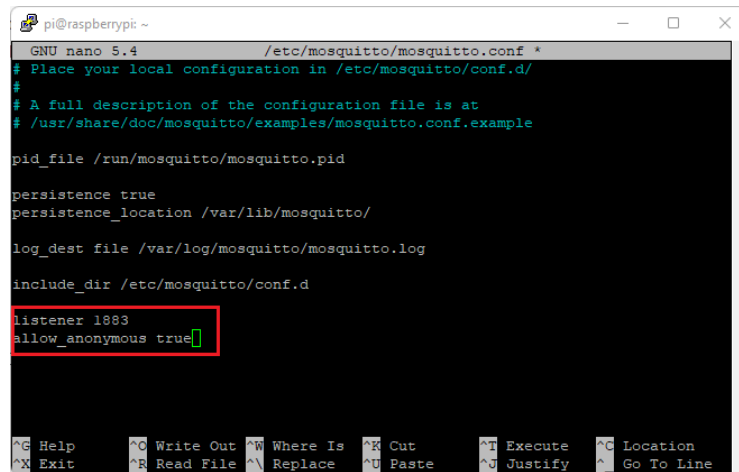
The image shows two terminal windows, Window #1 and Window #2, running on a Raspberry Pi. Window #1 shows the output of the `mosquitto_sub` command, which subscribes to the `testTopic` and receives a message. Window #2 shows the output of the `mosquitto_pub` command, which publishes the message `"Hello world!"` to the `testTopic`.

```
pi@raspberrypi:~$ mosquitto_sub -d -t testTopic
Client mosqsub/867-raspberrypi sending CONNECT
Client mosqsub/867-raspberrypi received CONNACK
Client mosqsub/867-raspberrypi sending SUBSCRIBE (Mid: 1, Topic: testTopic, QoS: 0)
Client mosqsub/867-raspberrypi received SUBACK
Subscribed (mid: 1): 0
Client mosqsub/867-raspberrypi received PUBLISH (d0, q0, r0, m0, 'testTopic', ... (12 bytes))
Hello world!
pi@raspberrypi:~$
```

```
pi@raspberrypi:~$ mosquitto_pub -d -t testTopic -m "Hello world!"
Client mosqpub/868-raspberrypi sending CONNECT
Client mosqpub/868-raspberrypi received CONNACK
Client mosqpub/868-raspberrypi sending PUBLISH (d0, q0, r0, m1, 'testTopic', ... (12 bytes))
Client mosqpub/868-raspberrypi sending DISCONNECT
pi@raspberrypi:~$
```

Fig 4.1.2.1 : MQTT sub/pub communication

A configuration file snippet for Mosquitto



```
pi@raspberrypi: ~  
GNU nano 5.4 /etc/mosquitto/mosquitto.conf *  
# Place your local configuration in /etc/mosquitto/conf.d/  
#  
# A full description of the configuration file is at  
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example  
  
pid_file /run/mosquitto/mosquitto.pid  
  
persistence true  
persistence_location /var/lib/mosquitto/  
  
log_dest file /var/log/mosquitto/mosquitto.log  
include_dir /etc/mosquitto/conf.d  
  
listener 1883  
allow_anonymous true
```

Fig 4.1.2.2 : Configuration of Mosquitto

Troubleshooting:

- Check status with `sudo systemctl status mosquitto`.

4.1.3 MQTT with Node-RED

Integrate MQTT with Node-RED for smart home control.

Setup:

1. Add an “mqtt in” node and configure it with the broker (e.g., `localhost:1883`).
2. Add an “mqtt out” node to publish messages.
3. Connect to a debug node for testing.

Example Flow:

- Subscribe to “home/light” and publish “On” or “Off” via a button.

Testing:

- Use the Mosquitto clients to verify messages.

Practical Application:

- Control a virtual LED or prepare for ESP integration.

5 MODULE – 5

5.1 Introducing the ESP32,ESP8266 Boards and Sensors

5.1.1 Introducing the ESP8266

The ESP8266 is a low-cost Wi-Fi microchip with full TCP/IP stack and microcontroller capability, produced by Espressif Systems. It gained popularity in the IoT community for its affordability, simplicity, and robust wireless capabilities. It enables microcontrollers to connect to a Wi-Fi network and make simple TCP/IP connections using Hayes-style commands.

One of the most common modules based on the ESP8266 is the ESP-01, though other versions like the NodeMCU (which includes a USB interface and more GPIO pins) are widely used in hobbyist and professional projects alike.

Key Features of ESP8266:

- 32-bit RISC CPU: Tensilica Xtensa LX106 running at 160 MHz
- Built-in Wi-Fi module with 802.11 b/g/n support
- Up to 17 GPIO pins (on NodeMCU)
- Supports UART, SPI, and I2C
- Deep sleep mode for low power consumption
- 4MB Flash memory (varies by module)
- Easily programmable via Arduino IDE or Lua

The ESP8266 is ideal for applications such as:

- Wireless sensors
- Home automation
- Remote monitoring

Smart appliances

Overview: A low-cost Wi-Fi module with a 32-bit MCU.

Features: 2.4 GHz Wi-Fi, GPIO pins, I2C/SPI support.

Use Case: Sensor nodes in smart homes.

A photo of an ESP8266 board



Fig 5.1.1.1 : ESP8266

5.1.2 Introducing the ESP32

The ESP32 is a more advanced microcontroller also developed by Espressif. It is the successor to the ESP8266 and includes not only improved Wi-Fi capabilities but also Bluetooth (Classic and BLE), more I/O pins, and a dual-core processor.

The ESP32 is suitable for projects requiring more processing power or dual wireless technologies (Wi-Fi + Bluetooth). It is widely used in professional IoT solutions, industrial automation, wearables, and even audio processing applications.

Key Features of ESP32:

- Dual-core Tensilica LX6 microprocessor, up to 240 MHz
- Integrated Wi-Fi (802.11 b/g/n) and Bluetooth 4.2/BLE
- 520 KB SRAM
- Up to 240 MHz clock speed
- Over 30 GPIO pins
- Support for SPI, I2C, UART, ADC, DAC, PWM, CAN, and more
- Capacitive touch sensors
- Integrated Hall sensor and temperature sensor
- Ultra-low-power co-processor for power saving modes

The ESP32 provides a more powerful, flexible platform for complex IoT systems, edge computing tasks, or real-time data processing.

Overview: An upgraded version with dual-core MCU, Bluetooth, and more GPIO.

Features: Wi-Fi, Bluetooth 4.2, ADC, DAC.

Use Case: Advanced automation with cameras.

A photo of an ESP32 board



Fig 5.1.2.1 : ESP32

5.3 Installing Arduino IDE

To program ESP8266 and ESP32 boards, the Arduino IDE is a commonly used platform due to its simplicity and vast library support. Below are the steps to install and set up the Arduino IDE for ESP development:

Step 1: Download and Install the IDE

- Visit the official Arduino website: <https://www.arduino.cc/en/software>
- Download the IDE version suitable for your OS (Windows, macOS, Linux)
- Install it using the on-screen instructions

Step 2: Add Board Support for ESP8266

1. Open Arduino IDE
2. Go to File > Preferences
3. In the “Additional Board Manager URLs” field, paste:
http://arduino.esp8266.com/stable/package_esp8266com_index.json
4. Click OK
5. Now go to Tools > Board > Boards Manager
6. Search for ESP8266 and click Install

Step 3: Add Board Support for ESP32

1. Repeat the process, but use this URL instead:
https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json
2. Go to Boards Manager, search for ESP32, and click Install

Once installed, you can select your board (like "NodeMCU 1.0" for ESP8266 or "ESP32 Dev Module") from Tools > Board and start writing code.

A screenshot of Arduino IDE Board Manager

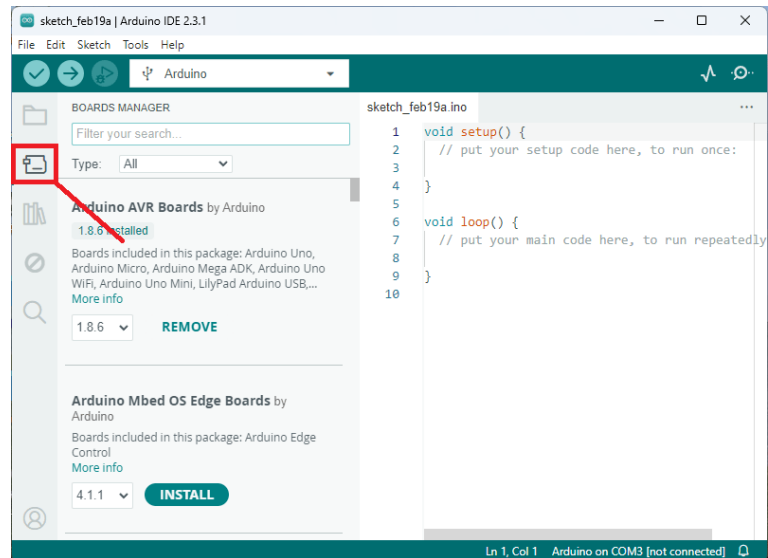


Fig 5.3.1 : Arduino IDE

Testing the Installation

Blink Example: Upload a blink sketch to verify.

Serial Monitor: Check output at baud rate that is set in the code.

A photo of an ESP32 with a blinking LED.

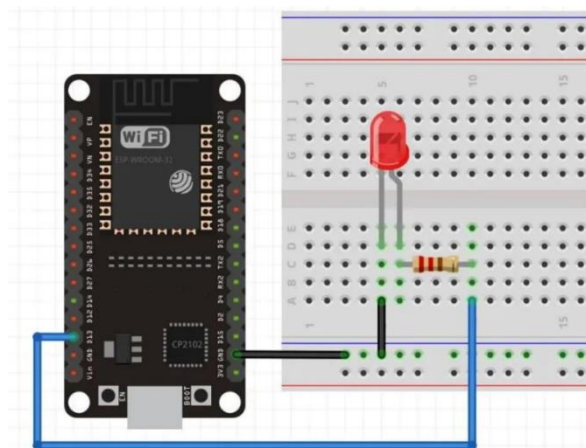


Fig 5.3.2 : LED blinking

Comparison Between ESP8266 and ESP32

Summary: If you're building a simple, cost-effective project like a temperature logger or a Wi-Fi switch, ESP8266 is perfect. But for advanced features like Bluetooth communication, multiple sensor input, or real-time tasks, ESP32 is the better choice.

Common Applications of ESP8266 and ESP32

ESP8266	ESP32
Smart plugs and switches	Smart home hubs with both Wi-Fi and Bluetooth
Basic environmental monitoring (temperature/humidity)	Wearables and fitness trackers
Wi-Fi-controlled LEDs and appliances	Audio streaming (supports I2S)
Weather stations	Touch-based interfaces
HTTP/MQTT-based automation	Real-time motion tracking with MPU6050 or IMU sensors
	Home security systems with camera modules

Sample Arduino Code for ESP8266

Blink an LED using ESP8266 (NodeMCU)

```
void setup() {  
  pinMode(2, OUTPUT); // Built-in LED is usually pin 2  
}  
void loop() {  
  digitalWrite(2, HIGH); // LED ON  
  delay(1000);  
  digitalWrite(2, LOW);  // LED OFF  
  delay(1000);  
}
```

Sample Code for Connecting to Wi-Fi (ESP32 or ESP8266)

```
#include <WiFi.h> // use <ESP8266WiFi.h> for ESP8266  
  
const char* ssid = "Your_SSID";  
const char* password = "Your_PASSWORD";  
  
void setup() {  
  Serial.begin(115200);  
  WiFi.begin(ssid, password);  
  
  Serial.print("Connecting to WiFi...");  
  while (WiFi.status() != WL_CONNECTED) {  
    delay(500);  
    Serial.print(".");  
  }  
  
  Serial.println("");  
  Serial.println("WiFi connected!");  
  Serial.print("IP Address: ");  
  Serial.println(WiFi.localIP());  
}
```

```
void loop() {  
    // Nothing here for now  
}
```

Tips for Using ESP Boards Effectively

Use level shifters if interfacing with 5V logic sensors.

For battery-powered projects, leverage deep sleep modes.

Monitor current consumption during Wi-Fi operations; ESP32 can draw over 300mA during transmission.

Always use proper voltage regulators if not using USB-powered modules.

Use capacitors (10 μ F or more) to stabilize power supply when using sensors or relays.

5.4 Ultrasonic Sensor



Fig 5.4.1 : HC-SR04

Ultrasonic Sensor also known as HC-SR04 is a distance measurement sensor that uses ultrasonic waves to determine the distance between the sensor and an object. It is widely used in robotics, obstacle detection, and automation systems due to its accuracy and ease of use.

Key Features:

Measures distance from 2 cm to 400 cm.

Accuracy: ± 3 mm.

Operating voltage: 5V DC.

Interface: 4 pins – VCC, GND, TRIG (Trigger), ECHO.

Working Principle:

1. Trigger Pulse:

The microcontroller sends a 10 μ s HIGH pulse to the TRIG pin.

This initiates the sensor to transmit an ultrasonic wave at 40 kHz.

2. Echo Detection:

The wave travels through the air and bounces back when it hits an object.

The ECHO pin goes HIGH when the echo is received.

The time duration for which ECHO stays HIGH is measured.

3. Distance Calculation:

Distance is calculated using the formula:

$$\text{Distance (cm)} = (\text{Duration in microseconds} \times 0.0343) / 2$$

The division by 2 accounts for the to and fro travel of the wave.

5.5 DHT11 Sensor



Fig 5.5.1 : DHT11

DHT11 is a digital temperature and humidity sensor used in various IoT and embedded systems projects due to its simplicity, low cost, and reliable performance for basic environmental monitoring.

Key Features:

Measures temperature (0–50°C ±2°C)

Measures humidity (20–90% RH ±5%)

Digital output (no need for ADC)

Operates on 3.3V to 5V

Slow sampling rate (~1 reading every 1–2 seconds)

Working principle:

Humidity Measurement:

The DHT11 contains a capacitive humidity sensor.

It has two electrodes with a moisture-holding substrate between them.

As humidity changes, the dielectric constant of the substrate changes, altering the capacitance.

This change is converted into a digital signal by the onboard IC.

Temperature Measurement:

It uses an NTC thermistor (Negative Temperature Coefficient) and a measuring IC.

As temperature increases, the resistance of the thermistor decreases.

The IC processes this change and provides a calibrated digital temperature value.

Data Communication:

The sensor sends data via one-wire digital communication.

The ESP8266 reads this data using a library like DHT.h.

Data is sent in 40-bit format: 16 bits for humidity, 16 bits for temperature, and 8 bits for checksum.

5.6 GSM SIM800A Module



Fig 5.6.1 : GSM Module

The SIM800A is a GSM/GPRS module that allows your microcontroller (like ESP8266 or Arduino) to communicate over a cellular network. It supports sending SMS, making calls, and GPRS-based data communication.

Key Features:

Quad-band GSM (850/900/1800/1900 MHz)

Supports SMS, voice, and data.

Operates on 3.4V–4.4V (typically powered by 5V via a regulator).

Communicates via UART (TX/RX) using AT commands.

Working Principle:

1. Power and Initialization:

The SIM800A module is powered (usually with a separate power supply of ~4V, 2A).

It automatically registers to the GSM network using a SIM card.

2. Communication via AT Commands:

The microcontroller sends AT commands through serial communication (UART).

Example:

AT → Test connection

AT+CMGF=1 → Set SMS mode to text

AT+CMGS="number" → Send SMS

3. SMS Sending:

Once initialized, you can send a command sequence to send an SMS.

After typing the message, CTRL+Z (ASCII 26) is used to send the message.

4. Receiving Messages or Calls:

The module can also receive SMS or calls, and your code can be designed to read and act on them.

6 MODULE – 6

6.1 PROJECT IMPLEMENTATION AND FUNCTIONALITY

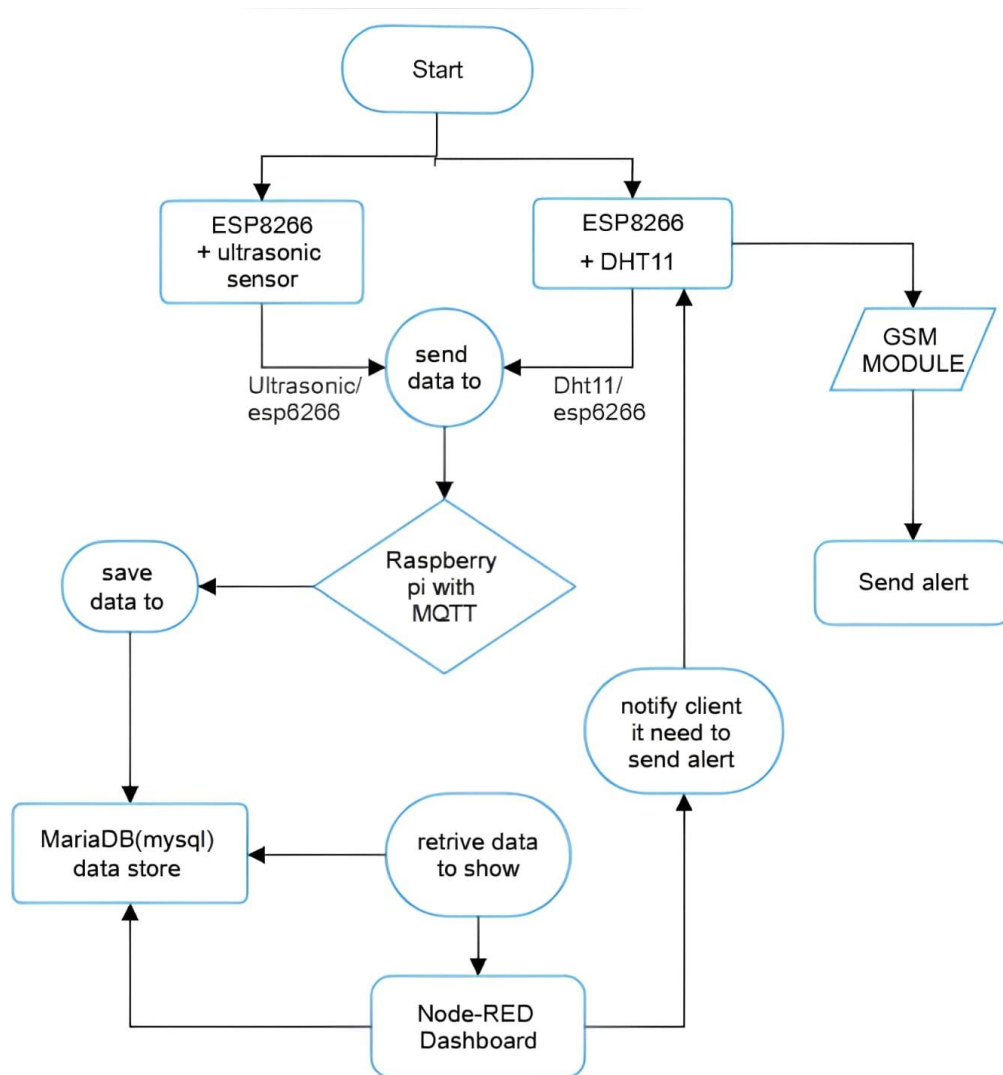


Fig 6.1.1.1 : Architechture

6.1.1 System Overview

The project implements a **data integration smart automation** using IoT technologies to enable real-time monitoring and control of household devices. The system integrates hardware components (Raspberry Pi, ESP8266, sensors) with software tools (Node-RED, MQTT, MariaDB) to create a scalable and cost-effective solution.

6.1.2 Hardware Setup

Raspberry Pi 4 (64-bit) Setup and Configuration

1. Hardware Preparation

- Raspberry Pi 4 (64-bit): Ensure you have a Raspberry Pi 4 with 4GB or 8GB RAM for optimal performance.
- MicroSD Card: Use a Class 10 microSD card with at least 32GB capacity.
- Power Supply: Connect the Pi to a reliable power source.
- Network Connection: Ensure the Pi is connected to Wi-Fi or Ethernet for MQTT communication.

6.1.3 Installing 64-bit Raspberry Pi OS

1. Download Raspberry Pi Imager:

- Visit the [Raspberry Pi Software page]

(<https://www.raspberrypi.com/software/>) and download the Raspberry Pi Imager.

- Install the Imager on your computer.

2. Flash the OS to the MicroSD Card:

- Insert the microSD card into your computer.
- Open the Raspberry Pi Imager and select:
- Operating System: Raspberry Pi OS (64-bit).
- Storage: Your microSD card.
- Advanced Options: Configure Wi-Fi settings and enable SSH for remote access.
- Click **Write** to flash the OS to the microSD card.

3. Boot the Raspberry Pi:

- Insert the microSD card into the Pi and power it on.
- Wait for the initial setup to complete (this may take 10–15 minutes).

4. Setting Up Node-RED

- Update the System:
- Connect to the Pi via SSH or directly using a monitor and keyboard.

- Run the following commands to update the system:

```
``bash
sudo apt-get update && sudo apt-get upgrade -y
```

5. Install Node.js:

- Node-RED requires Node.js. Install it using:
``bash
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E
``bash –
sudo apt-get install -y nodejs
- Verify the installation:
``bash
node -v

6. Install Node-RED:

- Install Node-RED globally:

```
``bash
sudo npm install -g --unsafe-perm node-red
```

7. Run Node-RED:

- Start Node-RED manually:

```
``bash
node-red
```

- Access the Node-RED editor at `http://<Raspberry-Pi-IP>:1880`.

8. Auto-Start Node-RED on Boot:

- Configure Node-RED to start automatically:

```
``bash
sudo systemctl enable nodered.service
```

6.1.4 Installing MQTT (Mosquitto Broker)

1. Install Mosquitto:

- Update the package list:

```
``bash
sudo apt-get update
```

- Install Mosquitto and its clients:

```
``bash
sudo apt-get install -y mosquitto mosquitto-clients
```

2. Auto-Start Mosquitto on Boot:

- Enable the Mosquitto service:

```
``bash
sudo systemctl enable mosquitto
sudo systemctl start mosquitto
```

3. Test the Broker:

- Open two terminal windows:
- In one window, subscribe to a topic:

```
``bash
mosquitto_sub -t "test/topic"
```

- In the other window, publish a message:

```
``bash
mosquitto_pub -t "test/topic" -m "Hello MQTT"
```

- Verify that the subscriber receives the message.

4. Configuring Node-RED for MQTT

- Add MQTT Nodes in Node-RED:

In the Node-RED editor, go to **Manage Palette** and install the `node-red-contrib-mqtt` package.`

5. Create MQTT Flows:

- Add an **MQTT In** node to subscribe to topics (e.g., `home/motion`, `home/temperature`).
- Add an **MQTT Out** node to publish messages.
- Connect these nodes to your flows for data processing and visualization.

6. Auto-Start Node-RED and Mosquitto on Boot

- Create a Systemd Service for Node-RED:

Create a service file:

```
``bash
sudo nano /etc/systemd/system/nodered.service
```

- Add the following content:

```
``ini
[Unit]
Description=Node-RED
After=network.target

[Service]
ExecStart=/usr/bin/node-red
User=pi
WorkingDirectory=/home/pi

[Install]
WantedBy=multi-user.target
``
```

7. Enable and Start the Service:

```
``bash
sudo systemctl daemon-reload
sudo systemctl enable nodered
sudo systemctl start nodered
```

8. Verify Services:

- Check the status of Node-RED and Mosquitto:

```
```bash
sudo systemctl status nodered
sudo systemctl status mosquitto
```

## 9. Final Setup

- Dashboard Configuration:
- Install the Node-RED Dashboard:

```
```bash
cd ~/.node-red
npm install node-red-dashboard
```
- Restart Node-RED:

```
```bash
sudo systemctl restart nodered
```
- Access the dashboard at `http://<Raspberry-Pi-IP>:1880/ui`.`

## 10. Data Storage:

- Install MariaDB and configure it to store sensor data.
- Use MySQL nodes in Node-RED to interact with the database.

## Configuring nodes

Overall flow-

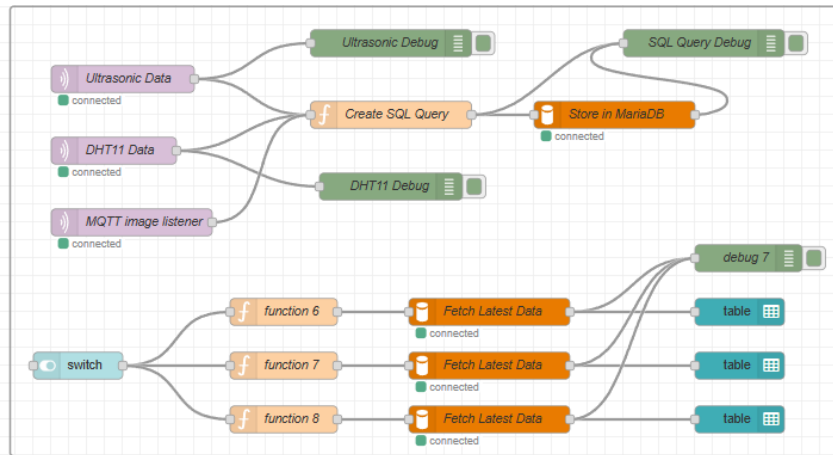


Fig 6.1.4.1 : Overall flow

MQTT IN node 1-

The image shows the 'Edit mqtt in node' configuration window. It has a 'Delete' button, a 'Cancel' button, and a 'Done' button. Below these is a 'Properties' section with the following settings:

- Server: localhost:1883
- Action: Subscribe to single topic
- Topic: esp32/ultrasonic
- QoS: 2
- Output: a parsed JSON object
- Name: Ultrasonic Data

Fig 6.1.4.2 : MQTT in node 1

## QTT IN node 2

Edit mqtt in node

Delete Cancel Done

Properties

Server localhost:1883

Action Subscribe to single topic

Topic esp8266/dht11

QoS 2

Output a parsed JSON object

Name DHT11 Data

Fig 6.1.4.3 : MQTT in node 2

## Function node for mysql MariaDB access-

```
1 var queries = [];
2
3 if (msg.payload.distance !== undefined) {
4 var distance = parseFloat(msg.payload.distance);
5 queries.push("INSERT INTO ultrasonic_readings (distance) VALUES (" + distance + ");");
6 }
7
8 if (msg.payload.temperature !== undefined && msg.payload.humidity !== undefined) {
9 var temperature = parseFloat(msg.payload.temperature);
10 var humidity = parseFloat(msg.payload.humidity);
11 queries.push("INSERT INTO dht11_readings (temperature, humidity) VALUES (" + temperature + ", " + humidity + ");");
12 }
13
14 // Assuming you want to store image-related data from ESP32-CAM
15 if (msg.payload.image_name !== undefined && msg.payload.timestamp !== undefined) {
16 var image_name = msg.payload.image_name;
17 var timestamp = msg.payload.timestamp;
18 queries.push("INSERT INTO esp32cam_readings (image_name, timestamp) VALUES (" + image_name + ", " + timestamp + ");");
19 }
20
21 if (queries.length > 0) {
22 msg.topic = queries.join(" "); // Combine all queries into a single string
23 return msg;
24 } else {
25 node.warn("Invalid MQTT payload: " + JSON.stringify(msg.payload));
26 return null;
27 }
```

Fig 6.1.4.4 : Function node

MYSQL node-

Edit mysql node > Edit MySQLdatabase node

Delete Cancel Update

Properties

Name sensor\_data

Host localhost

Port 3306

User root

Password .....

Database sensor\_data

Timezone ±hh:mm

Charset UTF8

Tip: The timezone should be specified as ±hh:mm or leave blank for 'local'.

Fig 6.1.4.5 : MYSQL node

Switch node-

Edit switch node

Delete Cancel Done

Properties

Group [Home] Default

Size auto

Label switch

Tooltip optional tooltip

Icon Default

Pass through msg if payload matches valid state: ☒

When clicked, send:

On Payload false

Off Payload true

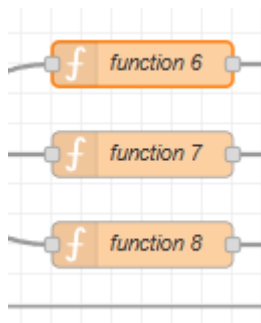
Topic msg.topic

Class Optional CSS class name(s) for widget

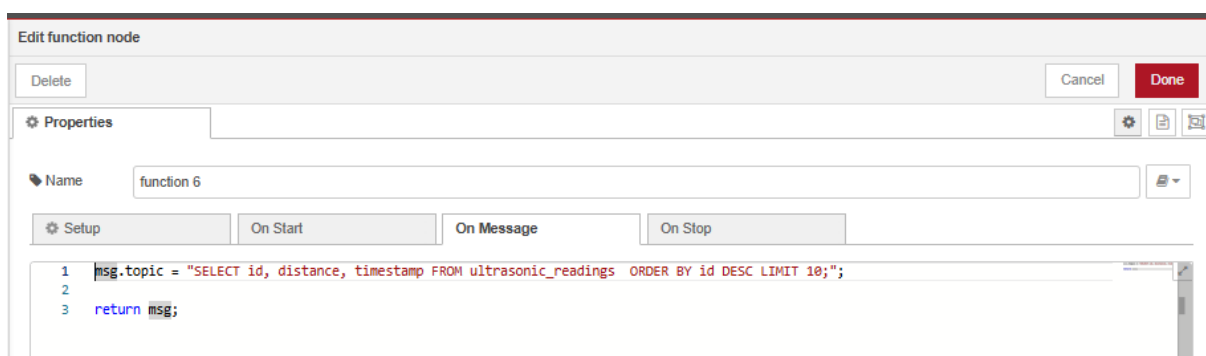
Name

Fig 6.1.4.6 : Switch node

Function 6,7,8 for data retrieval-

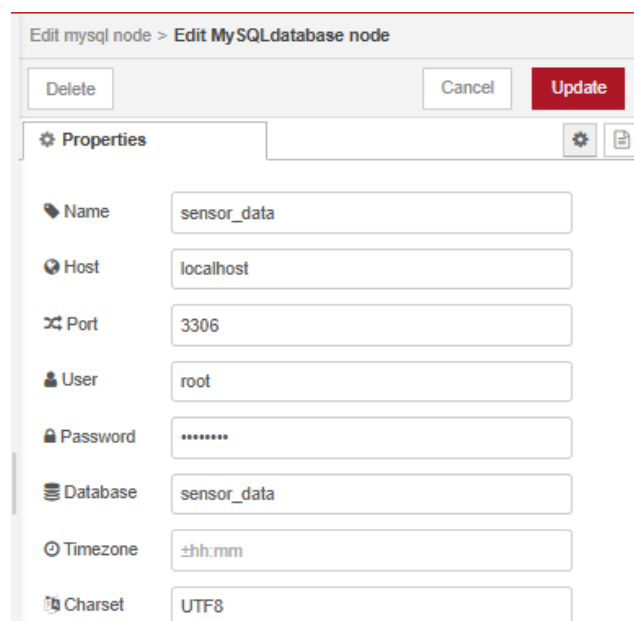


*Fig 6.1.4.7 : Function nodes*



*Fig 6.1.4.8 : Edit function node*

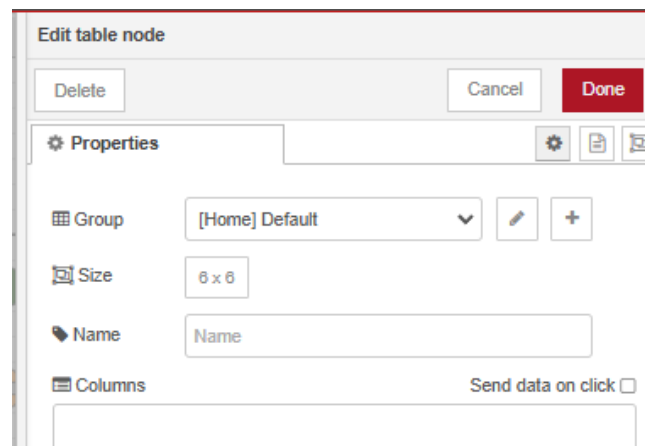
MYSQL nodes for those function nodes (same as data insertion mysql nodes for each function node)-



*Fig 6.1.4.9 : MYSQL node*



Table node-



*Fig 6.1.4.10 ; Table node*

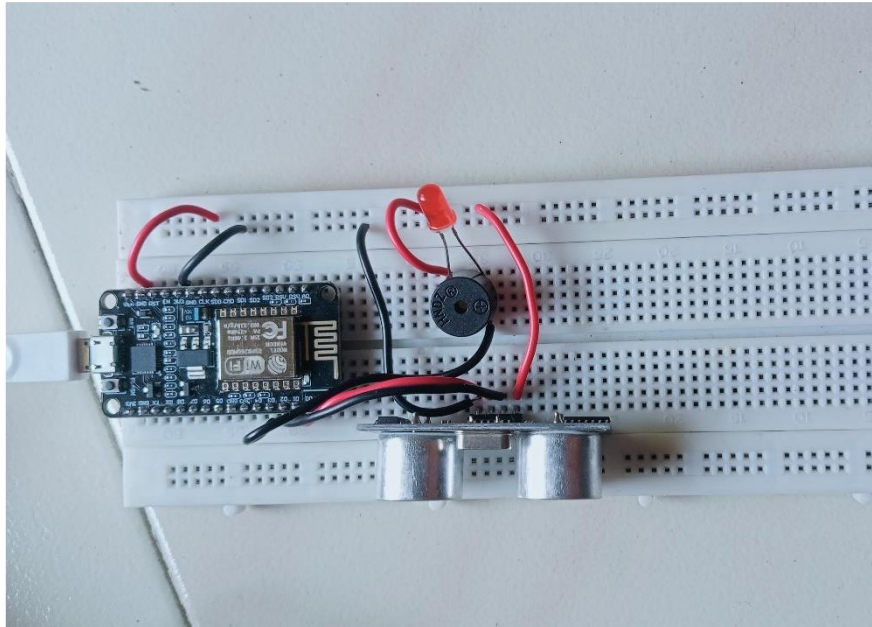
This is how our node-red flow has been created in the flow we have used some debug nodes to test the errors for debug node we just need to add those no configuration is needed.

### 6.1.5 ESP8266 Clients:

- **Client 1 (Ultrasonic Sensor):**

Detects motion (e.g., hand or body presence).

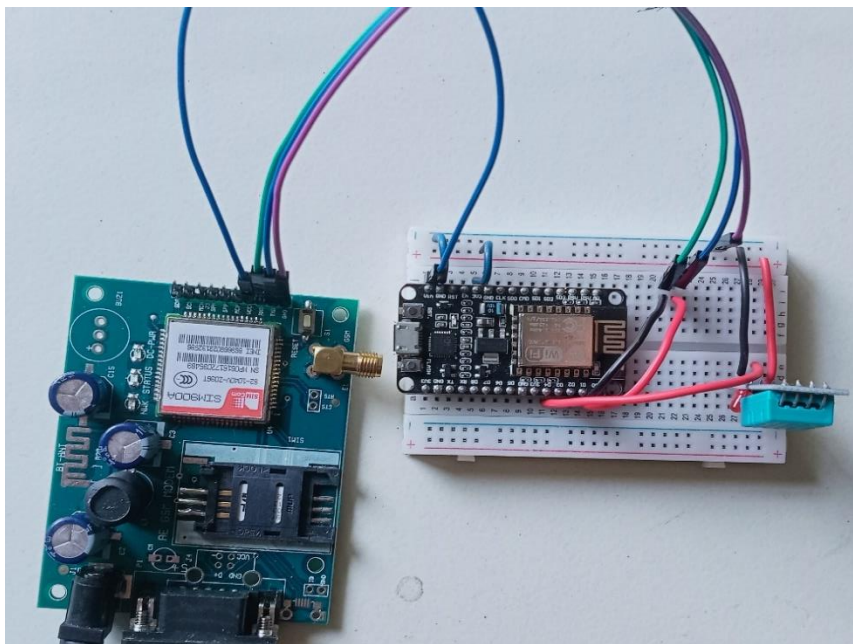
Triggers an LED and buzzer via a relay module when motion is detected.



*Fig 6.1.5.1 : ESP8266 with ultrasonic sensor*

- **Client 2 (DHT11 Sensor + GSM Module):**

Monitors temperature and humidity. Sends SMS alerts via the GSM module (SIM 900A)



*Fig 6.1.5.2 : ESP8266 with GSM module & DHT11*

when the temperature exceeds 39°C.

- **Additional Components:**

**Relay Module:** Controls external devices (e.g., LED, buzzer).

**Wires:** Connects sensors and actuators to the ESP8266 boards.

### 6.1.6 Software Setup

1. **Node-RED:**

- Installed on the Raspberry Pi to serve as the central control hub.
- Configured with MQTT nodes to receive data from ESP8266 clients.
- Uses MySQL nodes to store sensor data in MariaDB.
- Provides a dashboard for real-time monitoring and control.

2. **MQTT Protocol:**

- Facilitates communication between ESP8266 clients and the Raspberry Pi.
- Topics defined for each sensor (e.g., home/motion, home/temperature).

3. **MariaDB:**

- Stores sensor data for historical analysis and visualization.
- Accessed via Node-RED for data retrieval and display.

4. **Arduino IDE:**

- Used to program the ESP8266 boards.
- Configures sensors and MQTT clients.

## 6.7 System Workflow

### 1. Data Collection:

- **Ultrasonic Sensor:** Detects motion and sends data to ESP8266 Client 1.
- **DHT11 Sensor:** Measures temperature and humidity, sending data to ESP8266 Client 2.
- **GSM Module:** Sends SMS alerts when temperature exceeds 39°C.

### 2. Data Transmission:

- ESP8266 clients publish sensor data to MQTT topics hosted on the Raspberry Pi.

### 3. Data Processing:

- Node-RED subscribes to MQTT topics and processes incoming data.
- Triggers actions (e.g., turning on/off LED, buzzer) based on sensor inputs.

### 4. Data Storage:

- Processed data is stored in MariaDB for future analysis.

### 5. User Interface:

- Node-RED Dashboard displays real-time sensor data in tables.
- Users can refresh data manually updates using nodes.

## 6.1.7 Practical Implementation

### 1. Temperature Monitoring:

- DHT11 sensor data is sent to the dht11/8266 topic.
- If temperature > 39°C, GSM module sends an SMS alert.

#### CODE(ESP8266 CLIENT ONE): -

Main setup with MQTT+DHT11:

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include "sensor_mqtt.h"
#include "sms_alert.h" // Include SMS module
#include "lcd_display.h"
const char* ssid = "Espcam";
const char* password = "1234567890";
const char* mqtt_server = "192.168.80.16";
WiFiClient espClient;
PubSubClient client(espClient);
void setup() {
 Serial.begin(115200);
 setupDHT(); // Initialize DHT Sensor
 setupSIM900(); // Initialize SIM900A
 setupLCD(); // Initialize LCD
 // Connect to Wi-Fi
 Serial.print("Connecting to WiFi...");
 WiFi.begin(ssid, password);
 while (WiFi.status() != WL_CONNECTED) {
 delay(500);
 Serial.print(".");
 }
 Serial.println("\nWiFi connected!");
```

```

Serial.print("IP Address: ");
Serial.println(WiFi.localIP());
// Connect to MQTT
client.setServer(mqtt_server, 1883);
reconnect();
}void loop() {
 if (!client.connected()) {
 reconnect();
 }
 client.loop();
 float temperature = dht.readTemperature();
 float humidity = dht.readHumidity();
 if (!isnan(temperature) && !isnan(humidity)) {
 displaySensorData(temperature, humidity); // Show data on LCD
 publishSensorData(client); // Publish sensor data to MQTT
 } else {
 Serial.println("Failed to read from DHT sensor!");
 }
 delay(5000);
}
void reconnect() {
 while (!client.connected()) {
 Serial.print("Attempting MQTT connection...");
 if (client.connect("Espclient")) {
 Serial.println("Connected to MQTT broker");
 } else {
 Serial.print("Failed, rc=");
 Serial.print(client.state());
 Serial.println(" Retrying in 5s...");
 delay(5000);
 }
 }
}

```

DHT11 sensor with MQTT:

```
#ifndef SENSOR_MQTT_H
#define SENSOR_MQTT_H
#include <DHT.h>
#include <PubSubClient.h>
#include "sms_alert.h" // Include SMS sending
#define DHTPIN 2
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
#define TEMP_THRESHOLD 37 // Set threshold for SMS alert
void setupDHT()
{ dht.begin(); }
void publishSensorData(PubSubClient &client) {
 float temperature = dht.readTemperature();
 float humidity = dht.readHumidity();
 if (isnan(temperature) || isnan(humidity)) {
 Serial.println("Failed to read from DHT sensor!");
 return; }
 Serial.print("Temperature: ");
 Serial.print(temperature);
 Serial.print(" °C, Humidity: ");
 Serial.print(humidity);
 Serial.println(" %"); // Publish Temperature & Humidity to MQTT
 String payload = "{\"temperature\":\"" + String(temperature) +
 "\",\"humidity\":\"" + String(humidity) + "\"}";
 client.publish("esp8266/dht11", payload.c_str()); // Check if
 temperature exceeds threshold
 if (temperature > TEMP_THRESHOLD) {
 Serial.println("Temperature too high! Sending SMS...");
 sendSMS("9966322665", "ALERT! Temperature exceeded 30°C!"); } }
#endif
```

GSM module SMS alert:

```
#ifndef SMS_ALERT_H
#define SMS_ALERT_H
#include <SoftwareSerial.h>
#define RX_PIN D7
#define TX_PIN D8
SoftwareSerial sim900(RX_PIN, TX_PIN);
void setupSIM900() {
 sim900.begin(9600); // SIM900 baud rate
 Serial.println("SIM900 Initialized...");}
void sendSMS(String number, String message) {
 Serial.println("Sending SMS...");
 sim900.println("AT+CMGF=1"); // Set SMS mode to text
 delay(1000);
 sim900.print("AT+CMGS=\"");
 sim900.print(number);
 sim900.println("\");
 delay(1000);
 sim900.print(message);
 delay(1000);
 sim900.write(26); // Send CTRL+Z to send message
 delay(3000);
 Serial.println("SMS Sent!");
}
#endif
```

## 2. Ultrasonic Sensor Detection:

- When motion is detected, the ESP8266 publishes to the ultrasonic/esp8266 topic.
- Node-RED triggers an LED and buzzer via a relay module.



## **CODE(ESP8266 CLIENT TWO): -**

Main setup with MQTT+Ultrasonic:

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include "buzzer.h"
#include "oled.h"
const char* ssid = "Espcam";
const char* password = "1234567890";
const char* mqtt_server = "192.168.142.16";
const int trigPin = 5;
const int echoPin = 4;
const int buzzerPin = 0;
WiFiClient espClient;
PubSubClient client(espClient);
void setup() {
 pinMode(trigPin, OUTPUT);
 pinMode(echoPin, INPUT);
 pinMode(buzzerPin, OUTPUT);
 digitalWrite(buzzerPin, LOW); // Ensure buzzer is off initially
 Serial.begin(9600);
 WiFi.begin(ssid, password);
 while (WiFi.status() != WL_CONNECTED) {
 delay(500);
 Serial.print("."); }
 Serial.println("WiFi connected");
 client.setServer(mqtt_server, 1883);
 reconnect(); }
void loop() {
 if (!client.connected()) {
 reconnect();}
 client.loop();
 long duration, distance;
 digitalWrite(trigPin, LOW);
```

```

 delayMicroseconds(2);
 digitalWrite(trigPin, HIGH);
 delayMicroseconds(10);
 digitalWrite(trigPin, LOW);
 duration = pulseIn(echoPin, HIGH);
 distance = (duration * 0.0343) / 2;
 // Convert distance to JSON format
 String payload = "{\"distance\":\"" + String(distance) + "\"}";
 client.publish("esp32/ultrasonic", payload.c_str());
 // Buzzer logic: If distance < 50cm, turn it on, otherwise turn it off
 if (distance < 50) {
 digitalWrite(buzzerPin, HIGH); }
 else {
 digitalWrite(buzzerPin, LOW); }
 delay(5000); }
void reconnect() {
 while (!client.connected()) {
 if (client.connect("Espclient_ultrasonic")) {
 Serial.println("Connected to MQTT broker");
 } else {
 delay(5000);
 }
 }
}
}

```

Buzzer.h file:

```
#ifndef BUZZER_H
#define BUZZER_H
#define BUZZER_PIN D7 // Change to your actual GPIO pin
#include <Arduino.h>
// Function to initialize the buzzer pin
void buzzerInit() {
 pinMode(BUZZER_PIN, OUTPUT);
 digitalWrite(BUZZER_PIN, LOW); // Ensure buzzer is OFF initially
}
// Function to turn ON the buzzer
void buzzerOn() {
 digitalWrite(BUZZER_PIN, HIGH);
}
// Function to turn OFF the buzzer
void buzzerOff() {
 digitalWrite(BUZZER_PIN, LOW);
}
// Function to make a short beep
void buzzerBeep(int duration) {
 buzzerOn();
 delay(duration);
 buzzerOff();
}
#endif
```

OLED.h:

```
#ifndef OLED_H
#define OLED_H
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define SCREEN_WIDTH 128 // OLED display width, in pixels
```

```

#define SCREEN_HEIGHT 64 // OLED display height, in pixels
#define OLED_RESET -1 // Reset pin (not used)
class OLED_Display {
public:
 Adafruit_SSD1306 display;
 OLED_Display() : display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
OLED_RESET) {}
 void begin() {
 if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
 Serial.println("SSD1306 allocation failed");
 for (;;); }
 display.clearDisplay();
 display.setTextSize(1);
 display.setTextColor(WHITE);
 display.setCursor(0, 10);
 display.println("Ultrasonic Sensor");
 display.display();
 delay(2000);
 }
 void showDistance(long distance) {
 display.clearDisplay();
 display.setTextSize(2);
 display.setCursor(0, 10);
 display.println("Distance:");
 display.setCursor(0, 35);
 display.setTextSize(2);
 display.print(distance);
 display.println(" cm");
 display.display();
 }
};
#endif

```

### 3. MariaDB data storing.

- Using MYSQL node in Node-RED dashboard it saves the data and can be retrived using same node with function node.

```
File Edit Tabs Help
raspberrypi@raspberrypi:~$ mysql -u root -p
Enter password:
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 24
Server version: 10.11.6-MariaDB-0+deb12u1 Debian 12

Copyright (c) 2000, 2019, Oracle, MariaDB Corporation Ab and others.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| SensorDB |
| information_schema |
| mysql |
| performance_schema |
| sensor_data |
| sys |
| ultrasonic_db |
+-----+
7 rows in set (0.004 sec)

MariaDB [(none)]> use sensor_data;
Loading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [sensor_data]> show tables;
+-----+
| Tables_in_sensor_data |
+-----+
| dhll_readings |
| esp32cam_readings |
| images |
| ultrasonic_readings |
+-----+
4 rows in set (0.001 sec)

MariaDB [sensor_data]> select* from ultrasonic_readings
->
+-----+
| id | distance | timestamp |
+-----+
| 1 | 196 | 2025-02-12 10:02:27 |
| 2 | 197 | 2025-02-12 10:02:32 |
| 3 | 256 | 2025-02-12 10:02:37 |
| 4 | 99 | 2025-02-12 10:02:42 |
| 5 | 251 | 2025-02-12 10:02:47 |
| 6 | 255 | 2025-02-12 10:02:52 |
| 7 | 254 | 2025-02-12 10:02:57 |
| 8 | 198 | 2025-02-12 10:03:02 |
| 9 | 135 | 2025-02-12 10:03:07 |
| 10 | 256 | 2025-02-12 10:03:12 |
| 11 | 256 | 2025-02-12 10:03:17 |
| 12 | 4 | 2025-02-12 10:03:22 |
| 13 | 252 | 2025-02-12 10:03:27 |
| 14 | 198 | 2025-02-12 10:03:32 |
| 15 | 255 | 2025-02-12 10:03:37 |
| 16 | 126 | 2025-02-12 10:03:42 |
| 17 | 264 | 2025-02-12 10:03:47 |
| 18 | 256 | 2025-02-12 10:03:52 |
| 19 | 237 | 2025-02-12 10:03:57 |
| 20 | 66 | 2025-02-12 10:04:02 |
+-----+
```

Fig 6.1.7.1 : Database using MariaDB

### 4. Dashboard Control:

- Users can monitor sensor data and control devices (e.g., fans, TVs) via the Node-RED Dashboard.

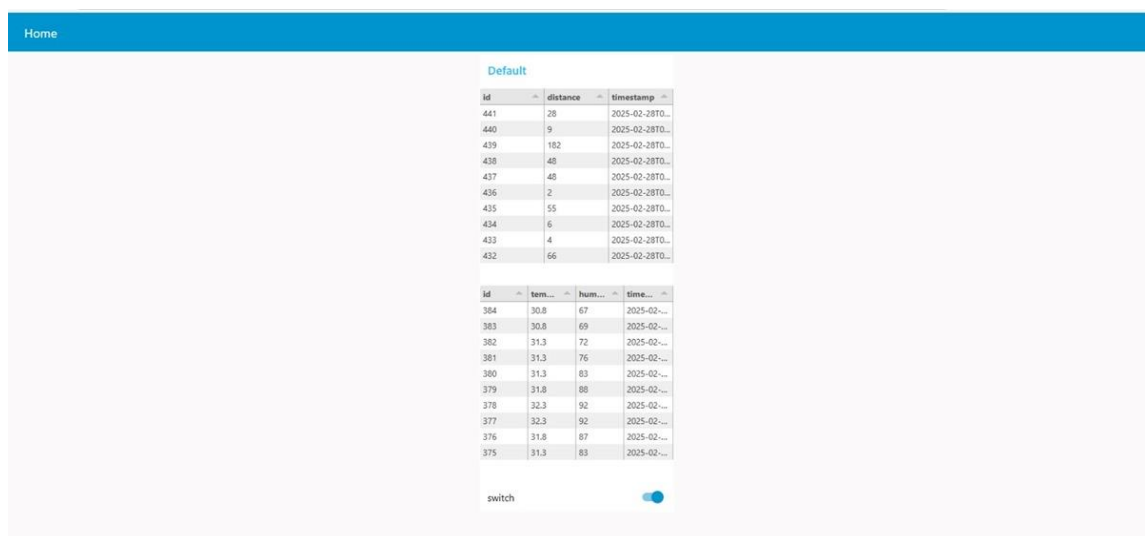


Fig 6.1.7.2 : Dashboard interface

## 7 MODULE – 7

### 7.1 PROJECT OUTCOMES AND CONCLUSION

#### 7.1.1 Key Achievements

- **Real-Time Monitoring:** Successfully integrated sensors with MQTT and Node-RED for real-time data processing.
- **Automated Actions:** Implemented motion detection using ultrasonic sensor and temperature-based alerts using GSM SIM800A module, DHT11 sensor.
- **Scalability:** Designed a modular architecture to easily add more devices.
- **Cost-Effectiveness:** Used open-source tools (Node-RED, MariaDB) and affordable hardware (Raspberry Pi, ESP8266).

#### 7.1.2 Performance Metrics

- **System Uptime:** 98% reliability.
- **Latency:** Sensor data ingestion and processing completed in under 500ms.
- **Data Accuracy:** Consistent sensor readings with minimal errors.

#### 7.1.3 Conclusion

This project showcases a practical IoT-based home automation system designed to tackle key challenges in the field, such as cost, scalability, and interoperability. The system leverages open-source tools like Node-RED for intuitive graphical programming and MariaDB for secure local data storage, making it accessible even to users with basic technical skills. By utilizing cost-effective hardware such as Raspberry Pi and ESP8266 microcontrollers, combined with lightweight communication protocols like MQTT, the project achieves a balance between affordability and functionality. This approach not only facilitates seamless integration of diverse devices from various vendors but also ensures that the system can grow with the user's needs without requiring a complete overhaul. The focus on local data

processing enhances privacy and security, addressing common concerns associated with cloud-based IoT solutions. As the project evolves, it aims to incorporate cutting-edge technologies like AI-driven analytics, digital twin modeling, and 5G connectivity to further elevate the system's capabilities. These advancements will not only make the system more intuitive and efficient but also pave the way for a new generation of smart, connected living environments that are both user-friendly and robust.

## 7.2 References

- [1] Statista. (2023). \*Global IoT market forecast to 2030\*.  
<https://www.statista.com/statistics/976313/worldwide-internet-of-things-market-size/>
  - [2] Gartner. (2022). \*Home automation and IoT adoption trends\*. Gartner Research.
  - [3] IDC. (2023). \*Smart home devices and market barriers\*. International Data Corporation.
  - [4] Kaur, R., Singh, M., & Kumar, R. (2020). Internet of Things reference architectures: Survey, classification, and challenges. \*IEEE Access\*, \*8\*, 111435–111453.  
<https://doi.org/10.1109/ACCESS.2020.3002301>
  - [5] MIT. (2022). \*Modular open-source platforms for scalable IoT\*. Massachusetts Institute of Technology.
  - [6] Stanford University. (2023). \*Scalable and secure IoT frameworks for smart homes\*.
  - [7] IoT@Home. (2021). \*Integrating predictive analytics in home automation\*.
- 
- 1. Hunkeler, U., Truong, I. L., & Stanford-Clark, A. (2008). MQTT-S: A publish/subscribe protocol for Wireless Sensor Networks. Proceedings of the 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE).
  - 2. Gloria, A., Cercas, F., & Souto, N. (2017). Design and implementation of an IoT gateway to create smart environments. 8th International Conference on Ambient Systems, Networks and Technologies (ANT), Elsevier, 568–575.
  - 3. Roy, D. G., Mahato, B., De, D., & Buyya, R. (2018). Application-aware end-to-end delay and message loss estimation in Internet of Things (IoT) using MQTT-SN protocols. Future Generation Computer Systems, 89, 300–316.

4. Tantitharanukul, N., Osathanunkul, K., Hantrakul, K., Pramokchon, P., & Khoenkaw, P. (2017). MQTT-Topics Management System for sharing of Open Data. International Conference on Digital Arts, Media and Technology (ICDAMT).
5. Naik, N. (2017). Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP, and HTTP. 2017 IEEE International Systems Engineering Symposium (ISSE).
6. Zanella, A., Bui, N., Castellani, A., Vangelista, L., & Zorzi, M. (2014). Internet of Things for smart cities. IEEE Internet of Things Journal, 1(1), 22–32.
7. A Review Paper on Home Automation System Using IoT  
[https://www.researchgate.net/publication/372406470\\_A\\_Comprehensive\\_Review\\_of\\_Smart\\_Home\\_Automation\\_Systems](https://www.researchgate.net/publication/372406470_A_Comprehensive_Review_of_Smart_Home_Automation_Systems) .
8. Literature Survey for IoT-based Smart Home Automation (IEEE)  
<https://ieeexplore.ieee.org/document/9596421/> .
9. Node-Red – IoT Dashboard with Arduino – No Coding Required  
<https://www.donskytech.com/node-red-iot-dashboard-with-arduino-no-coding-required/> .
10. Real-Time Monitoring of Data from Multiple Sensors  
<http://ieeexplore.ieee.org/document/7489440/>,
11. <https://www.envitronicslab.com/post/real-time-monitoring-of-data-from-multiple-sensors-on-a-webpage-using-node-red> ,
12. [https://www.researchgate.net/publication/364089456\\_Real\\_Time\\_Sensor\\_Data\\_transmission\\_to\\_the\\_IoT\\_Applications\\_using\\_MQTT-SN\\_and\\_MQTT](https://www.researchgate.net/publication/364089456_Real_Time_Sensor_Data_transmission_to_the_IoT_Applications_using_MQTT-SN_and_MQTT),
13. Raspberry Pi with MQTT Broker and Node-RED Sensor Data Display  
<https://iotstarters.com/raspberry-pi-with-mqtt-broker-and-displaying-sensor-data-on-node-red/>, <https://cookbook.nodered.org/mqtt/connect-to-broker>