COMPREHENSIVE HOME MONITERING SYSTEM WITH ENVIRONMENTAL SENSING USING ESP32

A Project Report Submitted and Partial Fulfilment of The Requirements for The Award of Degree Of

POLYTECHNIC

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by

M. PAVAN KUMAR	**K. PRASAD**	**A. YUVARAJ**
(21296-EC-052)	**(21296-EC-043)**	**(21296-EC-006)**

Under The Esteemed Guidance of

Ms. B. RADHA DEVI, M. Tech

Assistant Professor

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

CHAITANYA ENGINEERING COLLEGE

(Approved by AICTE) (Affiliated to SBTET, AP)

Madhurawada, Visakhapatnam- 530048, A.P

(2021-2024)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

CHAITANYA ENGINEERINF COLLEGE

(Approved By AICTE) (Affiliated To SBTET, AP)

Madhurawada, Visakhapatnam – 530048, A.P

<u>CERTIFICATE</u>

This is to certify that the project titled "**COMPREHENSIVE HOME MONITERING SYSTEM WITH ENVIRONMENTAL SENSING USING ESP32"** is a work done by **M. PAVAN KUMAR (21296-EC-052), K. PRASAD (21296-EC-043), A. YUVARAJ (21296-EC-006)** during the academic year 2021-2024 under our guidance. The work submitted to the "Department Of Electronics And Communication Engineering" in partial fulfillment for the award of the polytechnic in "Electronics And Communication Engineering".

Project guide	**Head of The Section**
Ms. B. RADHA DEVI, M. Tech	**Mrs. S. SIVALEELA, M. Tech**
Assistant professor	Assistant professor
ECE Department, CEC	ECE Department, CEC

ACKNOWLEDGEMENT

With great solemnity and sincerity, we offer our sincere thanks to management for providing all resources to complete our project successfully. We feel happy to record over sincere respect our principal **Dr. G. BHANU PRAVEEN, Ph. D** for his Valuable suggestions and encouragement while pursuing our study at college campus.

We express my sincere thanks to **Mrs. S. SIVALEELA, M. Tech** Lecturer in ECE, Chaitanya Engineering College, Head of the Department of Electronics and Communication Engineering for her Co-operation and persistent encouragement

We thank my guide **Ms. B. RADHA DEVI, M. Tech** for spending her valuable time to review and analyze my project at every stage. I consider myself extremely fortunate to have the opportunity of associating with her. We also thank full to all my faculty members, office staff and technical staff for their kind Co-operation.

Finally, our wish to take this opportunity to express our deep gratitude to all our friends who have extended their Co-operation in various ways during the project work. It is our pleasure acknowledges the help of all those individuals.

M. PAVAN KUMAR

(21296-EC-052)

K. PRASAD

(21296-EC-043)

A. YUVARAJ

(21296-EC-006)

ABSTRACT

This home monitoring system is like a smart helper for your house and plants. It uses a special ESP32 module to keep an eye on things like temperature, humidity, earthquakes, harmful gases, motion, and soil moisture. The information it collects is shown in real-time on a website and a small display in your home. This helps you understand what's happening in your house and take care of your plants better. The system is pretty smart! It can tell you if the temperature and humidity are just right for your comfort. It even warns you if there's an earthquake or if there are harmful gases around. The motion sensors help with security and let you know if someone's moving around. And if you have plants, the system makes sure they get just the right amount of water for healthy growth. The best part is that you can see all this information on a website from any device with the internet. Plus, there's a small display at home for quick updates. It's like having a helpful assistant to keep your home safe and your plants happy, thanks to the clever ESP32 technology.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBRIVATIONS

- LCD – Liquid Crystal Display
- HMS – Home Monitoring System
- ESP32 - digital temperature and humidity sensor
- TTL Logic - Transistor–transistor logic
- AO – Analog Output
- DO – Digital Output
- IR Flame Sensor – Infrared Flame Sensor
- LED – Light Emitting Diode
- SIG – Signal
- ULP – Ultra – Low Power
- RAM – Random Access Memory
- RTC – Real-Time Clock
- PWM – Pulse-Width Modulation
- UART - Universal Asynchronous Receiver-Transmitter
- GPIO - General-Purpose Input/Output
- ADC – Analog to Digital Converter
- HTML – Hyper Text Markup Language
- CSS - Cascading Style Sheets
- JS – Java Script
- HTTP - Hyper Text Transfer Protocol
- SMTP - Simple Mail Transfer Protocol

CHAPTER-1
INTRODUCTION

INTRODUCTON

1.1 INTRODUCTION

In today's fast-paced world, ensuring the safety and well-being of our homes has become paramount. The advent of smart technologies has empowered homeowners to monitor and secure their residences remotely. This project introduces a comprehensive Home Monitoring System (HMS) that utilizes the ESP32 microcontroller in conjunction with various sensors to provide real-time data on environmental conditions, security, and automation. By incorporating sensors such as gas sensors, DHT11 sensors, earthquake sensors, flame sensors, soil moisture sensors, and a water pump, along with a user-friendly web interface for data monitoring and security features like email alerts and home light indication, this system aims to offer a holistic solution to home monitoring and security needs.

1. Overview

The Home Monitoring System (HMS) is designed to offer homeowners a robust and user-friendly platform to monitor, control, and secure their homes remotely. The integration of the ESP32 microcontroller serves as the backbone of the system, facilitating seamless communication between sensors, actuators, and the user interface. By harnessing the power of various sensors, the HMS provides comprehensive insights into environmental parameters such as temperature, humidity, gas presence, earthquake detection, flame detection, and soil moisture levels. Additionally, the inclusion of a water pump enables automated irrigation based on soil moisture levels, enhancing the system's utility. Moreover, the implementation of a web-based interface allows users to monitor real-time data and receive email alerts for critical events, ensuring timely responses to potential threats or emergencies.

Furthermore, the integration of security features such as light indication in the home adds an extra layer of protection against intrusions.

The "Home Monitoring System" is an innovative project designed to enhance home safety and comfort using user-friendly technology. Leveraging the powerful ESP32 microcontroller and an array of sensors including the MQ2 for gas detection, SW-420 for vibrations, flame sensor for fire safety, and DHT-11 for temperature and humidity monitoring, this system acts as a vigilant guardian for your living space. The sensors collaborate seamlessly to provide real-time insights into potential hazards, ensuring a secure and pleasant environment.

Taking a step further into the digital realm, we've created a user-friendly website that allows you to remotely monitor your home's conditions. Whether it's checking gas levels, sensing vibrations, or keeping an eye on temperature, the information is at your fingertips, providing you with a sense of control and peace of mind no matter where you are. Additionally, the integration of a 16x2 LCD display provides a tangible interface for quick and easy updates on essential information. More than just a technological endeavor, the "Home Monitoring System" is a commitment to making homes safer and smarter, inviting you to join us in redefining the standards of home security and convenience for a better and connected future.

2. System Components

ESP32 Microcontroller: The ESP32 microcontroller serves as the central processing unit of the Home Monitoring System. Its dual-core architecture, coupled with built-in Wi-Fi and Bluetooth capabilities, enables efficient data processing and wireless communication with sensors, actuators, and the user interface.

Sensors:

Gas Sensor: Detects the presence of hazardous gases such as carbon monoxide (CO) or methane (CH4), providing early warnings to prevent potential accidents.

DHT11 Sensor: Measures temperature and humidity levels, helping users maintain optimal indoor conditions for comfort and health.
Earthquake Sensor: Detects seismic activity and provides alerts to mitigate risks and ensure the safety of occupants.

Flame Sensor: Identifies the presence of flames, enabling rapid response to fire emergencies and preventing property damage.

Soil Moisture Sensor: Monitors soil moisture levels to facilitate efficient irrigation and plant care, conserving water resources and promoting healthy vegetation.

Water Pump: Actuates irrigation systems based on soil moisture data, automating the process of watering plants.

3. User Interface and Security Features

Web-Based Interface: The HMS features a user-friendly web interface accessible from any internet-enabled device. Users can view real-time data from sensors, control actuators, and configure system settings remotely.

Email Alerts: The system sends email alerts to users in case of critical events such as gas leaks, earthquakes, or fire incidents, ensuring timely response and intervention.

Light Indication: Integrated light indication in the home provides visual alerts in case of security breaches or system anomalies, enhancing situational awareness and deterring intrusions.

Email Login: Users can securely access the web interface using email-based authentication, safeguarding sensitive data and ensuring authorized access to system controls and settings.

The Home Monitoring System presented in this project offers a comprehensive solution for monitoring and securing residential premises using the ESP32 microcontroller and a range of sensors. By integrating environmental monitoring, automation, and security features with a user-friendly web interface, the system empowers homeowners to remotely manage their homes and respond effectively to potential threats or emergencies. With its versatility, scalability, and emphasis on user experience and safety, the HMS represents a significant step towards creating smarter, safer, and more connected homes.

CHAPTER-2

SYSTEM MODELLING AND DESIGN

2.1 PROBLEM DEFINITION

Our project integrates eight circuits, including ESP-32, DHT-11, flame, MQ2 gas, LCD display, water pump, soil moisture, and SW-420 vibration sensors, optimizing their collaboration for wireless information sharing. Wired connections play a crucial role in ensuring efficient communication among devices. We aim to explore and address challenges in data transfer to enhance overall system performance. The goal is to create a seamless network where sensors work together, utilizing both wireless and wired mediums effectively.

Here, ESP32 acts as a heart of our project, in the above block diagram we can see that there are DHT-11, flame, MQ2 gas, LCD display, water pump, soil moisture, and SW-420 vibration sensors which acts as input interface to the micro controller and web interface, 16*2 lcd display, water pump acts a output interface to the micro controller, here the input and output interface can be indicated with the arrow lines with the respective the micro controller performs with the respective commands and delay which is programmed on Arduino IDE software

2.2 FUNCTIONAL DESCRIPTION

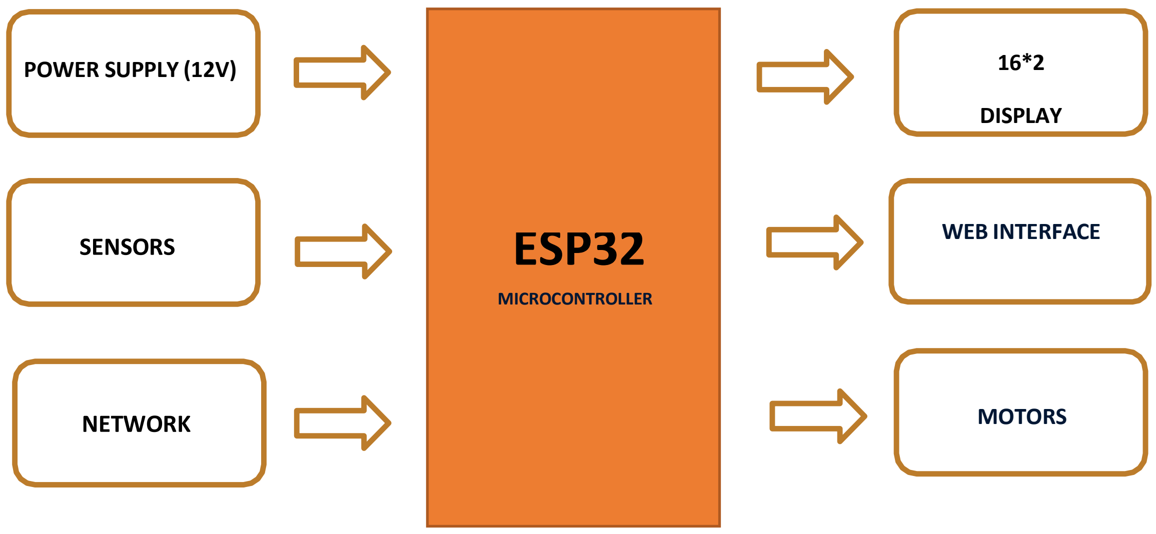


Fig 2.2: Functional Description of the System

The constituent parts involved in the process are

1. Sensors
2. ESP32 with Xtensa LX6 processor
3. Network
4. 16*2 display
5. Web interface
6. Motors
7. Power supply (12v)

First block portrays to be sensors which receives, verifies and forwards the message to the Micro-controller. The micro – controller process the signal and transmit the data to the web interface through the esp32 internal web server and display the data in 16*2 lcd display and web interface if any sensor value greater than the threshold value then the alert message is sent to the user and nearby police, fire and hospital if any emergency through email and also monitor the plant health and display the data to user in web interface and 16*2 lcd display

2.3 DATA FLOW DIAGRAM

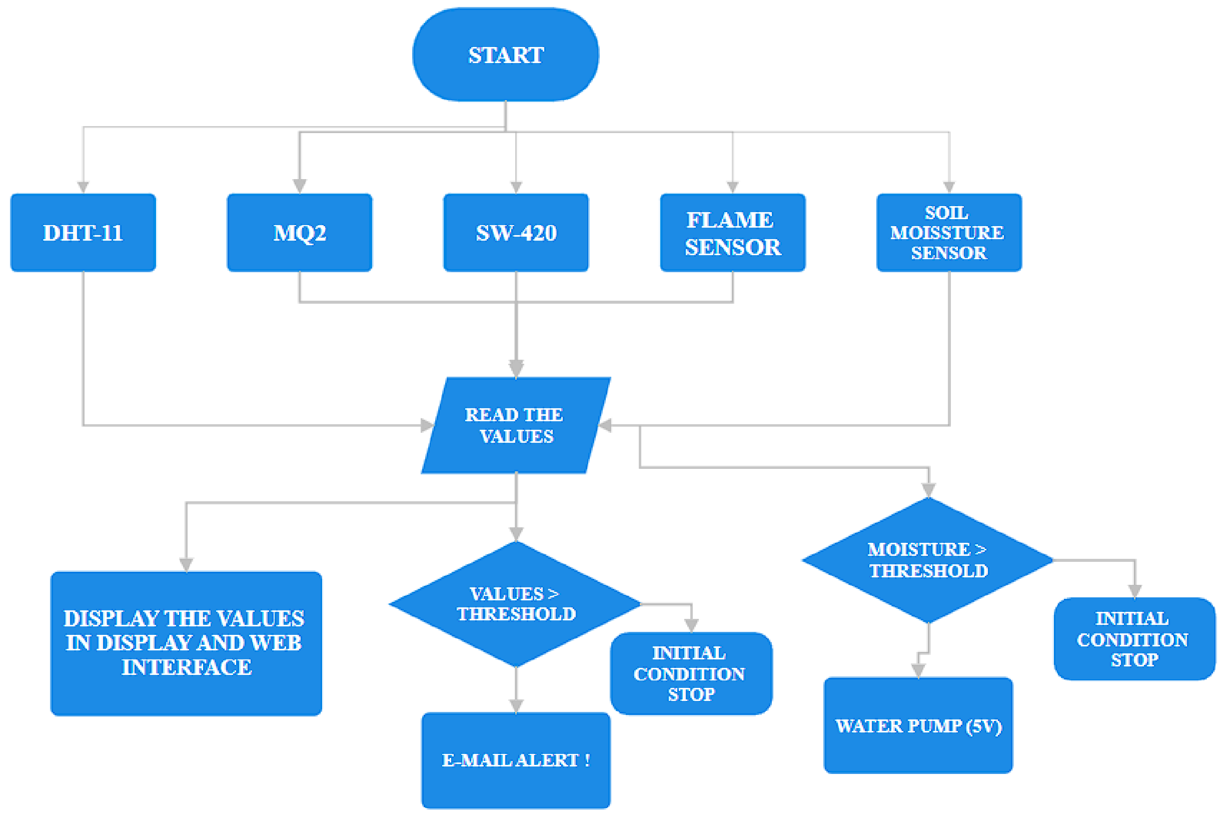


Fig 2.3: Data Flow Diagram of the System

The process begins with the initialization of component ports. Initially, the power supply is activated to energize the circuit. Five sensors are strategically positioned throughout the house to detect fluctuations in temperature, gas levels, seismic activity, and fire outbreaks. Should any of these sensors detect values surpassing predefined thresholds, an email alert is promptly dispatched. Simultaneously, the pertinent data is relayed and showcased on both a web interface and a 16x2 LCD display.

Furthermore, if the soil moisture sensor detects a moisture level exceeding the set threshold, the water pump is activated to initiate the watering process for the plants. This comprehensive system ensures timely detection and response to environmental changes within the household, enhancing safety and facilitating efficient plant care.

CHAPTER -3

HARDWARE SYSTEM REQUIREMENTS

3.1 HARDWARE COMPONENTS

3.1.1 MQ2 GAS SENSOR

The MQ2 gas sensor is a small electronic device designed to detect various gases in the air. It operates by measuring the concentration of gases such as methane, propane, carbon monoxide, and smoke

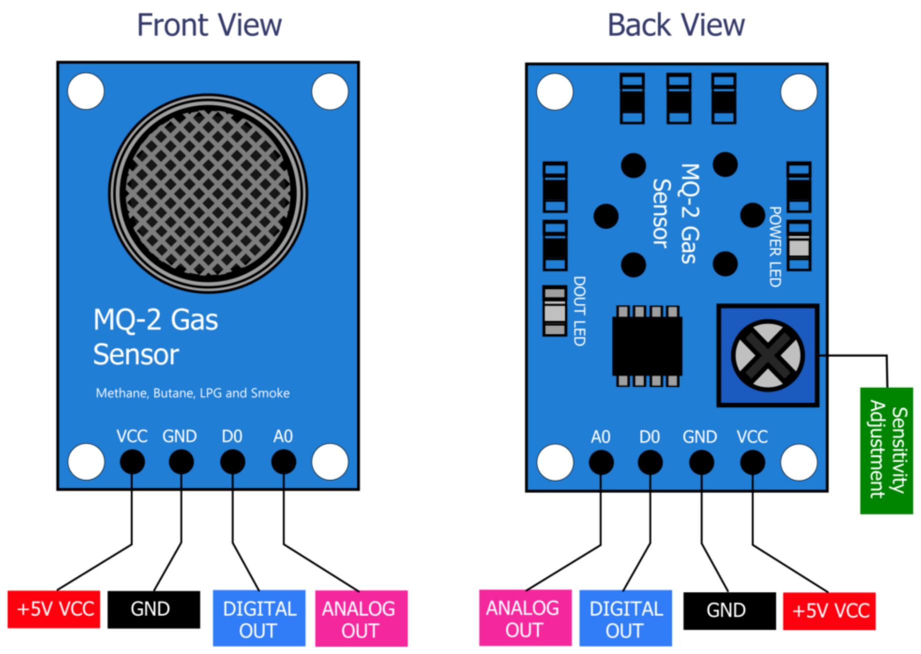


Fig 3.1.1 MQ2 gas sensor module

This sensor is commonly used in applications like gas leakage detectors, smoke alarms, and air quality monitoring systems. The MQ2 sensor works by changing its electrical resistance in response to the presence of different gases, and this change is then converted into a signal that can be interpreted by a microcontroller or other electronic devices. Its compact size and sensitivity make it a valuable tool in ensuring safety and maintaining air quality in various environments.

3.1.1.1 Features of MQ2 Gas Sensor

Hardware Features

1. Operating Voltage is +5V
2. Can be used to Measure or detect LPG, Alcohol, Propane, Hydrogen, CO and even methane
3. Analog output voltage: 0V to 5V
4. Digital Output Voltage: 0V or 5V (TTL Logic)
5. Preheat duration 20 seconds
6. Can be used as a Digital or analog sensor
7. The Sensitivity of Digital pin can be varied using the potentiometer

3.1.1.2 Pin Description

The MQ2 gas sensor features essential pins for seamless integration into electronic systems. The VCC pin connects to the power source, typically 5V, supplying energy to the sensor. The GND pin establishes the ground connection, ensuring a complete circuit and proper sensor functionality. The AO (Analog Output) pin produces an analog signal proportionate to the detected gas concentration, facilitating communication with microcontrollers. The DO (Digital Output) pin provides a digital signal that switches between high and low states based on predetermined gas concentration thresholds. Additionally, the sensor includes a heater element, connected to the power source, which maintains a consistent temperature for the gas-sensitive component, ensuring reliable and accurate gas detection.

Pin Number	Pin Name	Function
1	Vcc (+5v)	This pin powers the module, typically the operating voltage is +5V
2	Ground (GND)	Used to connect the module to system ground
3	Digital Out	You can also use this sensor to get digital output from this pin, by setting a threshold value using the potentiometer
4	Analog Out	This pin outputs 0-5V analog voltage based on the intensity of the gas

Table 3.1.1.2 Pin Description of MQ2 gas sensor

3.1.2 DHT-11 SENSOR

The DHT-11 sensor is a straightforward yet efficient device designed for measuring temperature and humidity in the surrounding environment. With its compact size and ease of use, the DHT-11 has become a popular choice in various applications, such as weather stations, home automation systems, and climate control devices. This sensor employs a digital output to relay temperature and humidity data, simplifying integration with microcontrollers like Arduino. Its affordability and reliability make the DHT-11 a practical solution for individuals and hobbyists looking to monitor and control the indoor climate conditions with ease.

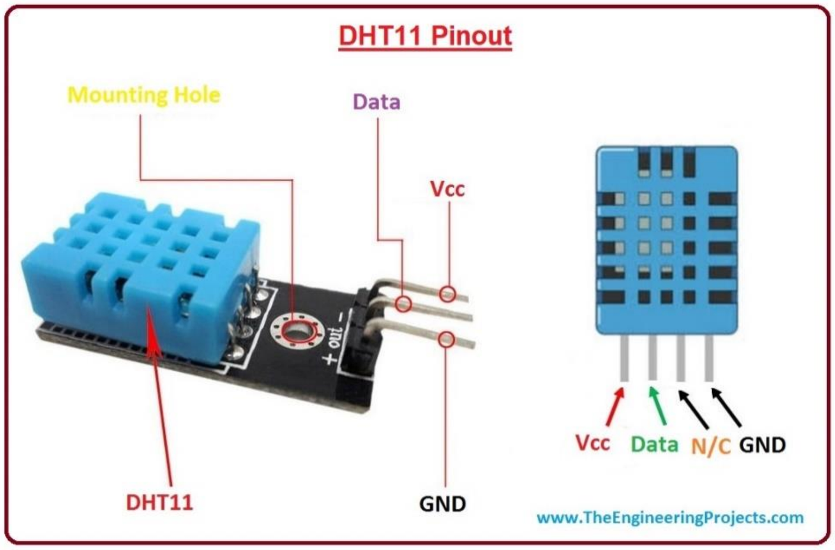


Fig 3.1.2 DHT-11 Sensor Module

The DHT-11 sensor operates by using a calibrated digital signal output, making it user-friendly for those with limited electronics experience. It utilizes a humidity-sensitive component and a thermistor to measure the respective parameters accurately. With its low cost and uncomplicated design, the DHT-11 sensor enables enthusiasts and developers to incorporate temperature and humidity sensing capabilities into their projects without extensive technical knowledge, contributing to the accessibility of environmental monitoring in a variety of applications.

3.1.2.1 Features of DHT-11 Sensor

1. Operating Voltage: 3.5V to 5.5V
2. Operating current: 0.3mA (measuring) 60uA (standby)
3. Output: Serial data
4. Temperature Range: 0°C to 50°C
5. Humidity Range: 20% to 90%

6. Resolution: Temperature and Humidity both are 16-bit
7. Accuracy: ±1°C and ±1%

3.1.2.2 Pin Description

The DHT-11 sensor features four essential pins for its operation. The VCC pin connects to the power source, typically at 3.3V or 5V, providing the necessary electrical energy. The Data Out pin is crucial, as it outputs a digital signal containing temperature and humidity data that can be read by a microcontroller or similar device. The third pin, labeled NC (Not Connected), is not utilized in standard applications and can be left unconnected. Ground (GND) is the fourth pin, requiring connection to the ground (0V) to complete the electrical circuit and ensure proper functionality. Additionally, for stability, it's common to include a resistor (usually 5k ohms) between the VCC and Data Out pins. This resistor, known as a pull-up resistor, helps maintain a reliable signal transmission between the sensor and the connected electronics.

Pin Number	Pin Name	Function
1	Vcc	Power supply 3.5V to 5.5V
2	Data	Outputs both Temperature and Humidity through serial Data
3	Nc	No Connection and hence not used
4	Ground	Connected to the ground of the circuit

Table 3.1.2.1 Pin Description of Dht-11 Sensor

3.1.3 Flame Sensor Module

Flame Sensor Module Fire Sensor Module infrared Receiver Ignition source detection module This tiny Flame sensor infrared receiver module ignition source detection module is Arduino compatible. It can use to detect flame or wavelength of the light source within 760nm~1100nm also useful for Lighter flame detection at a distance of 80cm. Greater the flame, the farther the test distance.

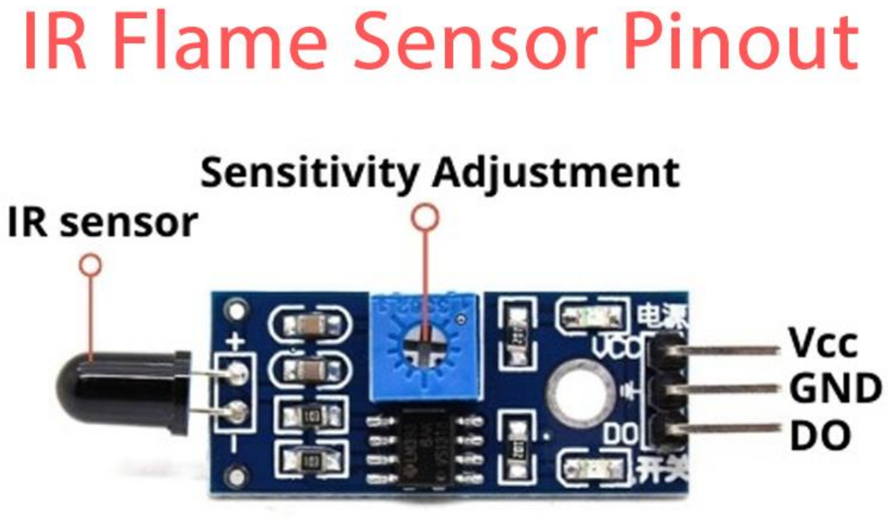


Fig 3.1.3 Ir Flame Sensor Module

It has a Detect angle of 60 degrees and is very sensitive to the flame spectrum. It produces the one-channel output signal at the D0 terminal for further processing like an alarm system or any switching system. The sensitivity is adjustable with the help of a blue potentiometer given on the board.

3.1.3.1 Features of IR Flame Sensor Module

1. Built-in a Potentiometer for Sensitivity Control.
2. Onboard Signal Output Indication, Output effective signal is high, and at the same time the indicator lights up, and the output signal can directly connect to the microcontroller IO.
3. Can detect fire or wavelength in 760 ~ 1100 nm nano within the scope of the light source.
4. The Detection Angle is about 60 degrees, and the flame spectrum is especially sensitive
5. The flame of the most sensitive sensors flame, the steady light is also a response, generally used for fire alarm purposes.

3.1.3.2 Pin Description

The IR Flame Sensor has three main pins for connection. The VCC pin is where you connect the power, usually 5V. The GND pin is for the ground connection. The DO pin provides a digital output signal that indicates the presence or absence of a flame. Additionally, there's an AO pin for analog output if you need a continuous measure of the flame intensity. Keeping the wiring simple, these pins enable easy integration with microcontrollers or other devices for flame detection applications.

Pin Number	Pin Name	Function
1	Vcc	+5 v power supply
2	Gnd	Ground (-) power supply
3	Out	Digital Output (0 or 1)

Table 3.1.3.1 Pin Description of Flame Sensor

3.1.4 SW-420 (Vibration Sensor)

This Vibration Sensor Module consists of an SW-420 Vibration Sensor, resistors, capacitor, potentiometer, comparator LM393 IC, Power, and status LED in an integrated circuit. It is useful for a variety of shocks triggering, theft alarm, smart car, an earthquake alarm, motorcycle alarm, etc.

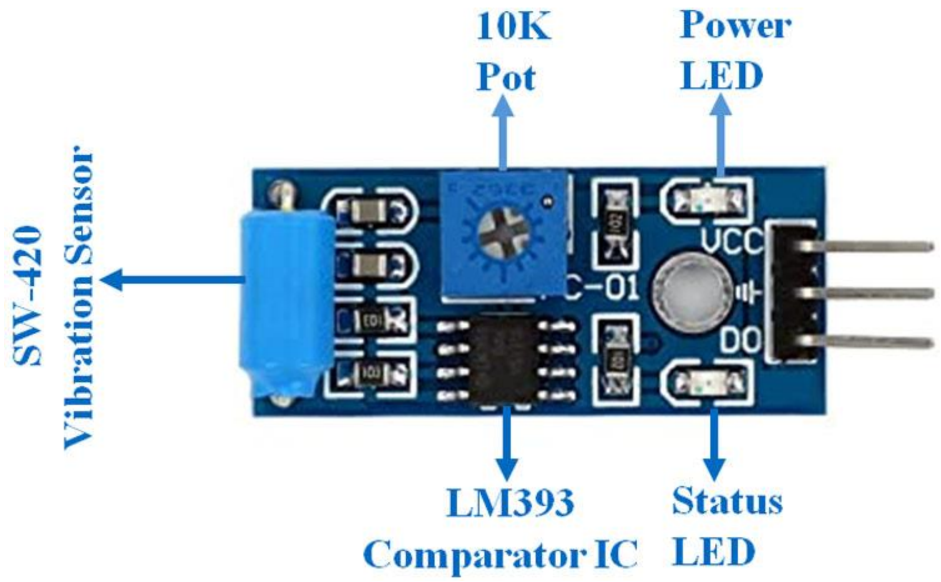


Fig 3.1.4 : SWW-420 Vibration Sensor Module

3.1.4.1 Features of SW-420 (Vibration Sensor)

1. Operating Voltage: 3.3V to 5V DC

2. Operating Current: 15mA
3. Using SW-420 normally closed type vibration sensor
4. LEDs indicating output and power
5. LM393 based design
6. Easy to use with Microcontrollers or even with normal Digital/Analog IC
7. With bolt holes for easy installation
8. Small, cheap and easily available

3.1.4.2 Pin Description of SW-420 (Vibration Sensor)

The SW-420 vibration sensor has three pins: VCC, GND, and SIG. Connect VCC to the power supply (typically 5V), GND to the ground, and SIG to a digital input pin on a microcontroller. When vibrations are detected, the SIG pin sends a digital signal (HIGH) to indicate motion. Adjust sensitivity by tweaking the potentiometer on the sensor, allowing for customization in detecting various vibration levels. Its straightforward three-pin setup makes it easy for beginners to integrate into projects requiring vibration sensing.

Pin Number	Pin Name	Function
1	Vcc	The Vcc pin powers the module, typically with +5V
2	Gnd	Power Supply Ground
3	D0	Digital Out Pin for Digital Output.

Table 3.1.4.2 Pin Description of SW-420 (Vibration Sensor)

3.1.5 16x2 LCD

we know that each character has (5×8=40) 40 Pixels and for 32 Characters we will have (32×40) 1280 Pixels. Further, the LCD should also be instructed about the Position of the Pixels. Hence it will be a hectic task to handle everything with the help of MCU, hence an Interface IC like HD44780 is used, which is mounted on the backside of the LCD Module itself. The function of this IC is to get the Commands and Data from the MCU and process them to display meaningful information onto our LCD Screen. You can learn how to interface an LCD using the above mentioned links. If you are an advanced programmer and would like to create your own library for interfacing your Microcontroller with this LCD module then you have to understand the HD44780 IC working and commands which can be found its datasheet.

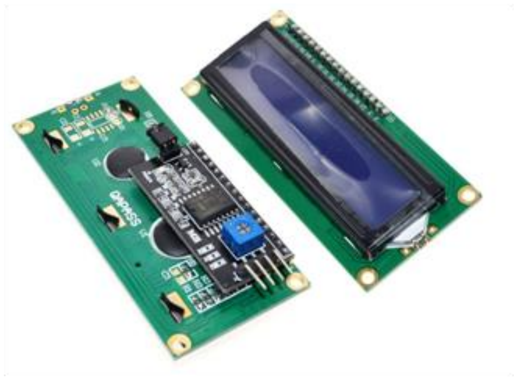


Fig 3.1.5 : 16*2 LCD Display

3.1.5.1 Features f 16x2 LCD With I2C

1. I2C Address Range 0x20 to 0x27 (Default=0x27, addressable)
2. Operating Voltage 5 Vdc
3. Backlight: White
4. Contrast: Adjustable by potentiometer on I2c interface

5. Size: 80mm x 36mm x 20 mm
6. Viewable area: 66mm x 16mm

3.1.5.2 Pin Description

The 16x2 LCD with I2C module has four pins: VCC for power, GND for ground, SDA for data transmission, and SCL for clock signal. The VCC pin connects to the power source, GND to the ground, SDA to data, and SCL to the clock line. The I2C module simplifies connections by allowing communication with the LCD using only two wires (SDA and SCL), streamlining the wiring process for easy integration with microcontrollers like Arduino.

Pin Number	Pin Name	Function
1	Gnd	Supply & Logic ground
2	Vcc	Power supply of +5v
3	SDA	Serial Data Line
4	SCL	Serial Clock Line

Table 3.1.5.2 Pin Description of I2C Module

3.1.6 Soil Moisture Sensor

FC-28 soil moisture sensor is a soil hygrometric transducer that can read the amount of moisture present in the soil surrounding it. The module uses the two probes to pass current through the soil, and then it reads that resistance to get the moisture level. More water makes the soil conduct electricity more easily (less resistance), while dry soil conducts electricity poorly (more resistance). FC-28 offer enhanced protection features such as ESD protection and under voltage protection feature.

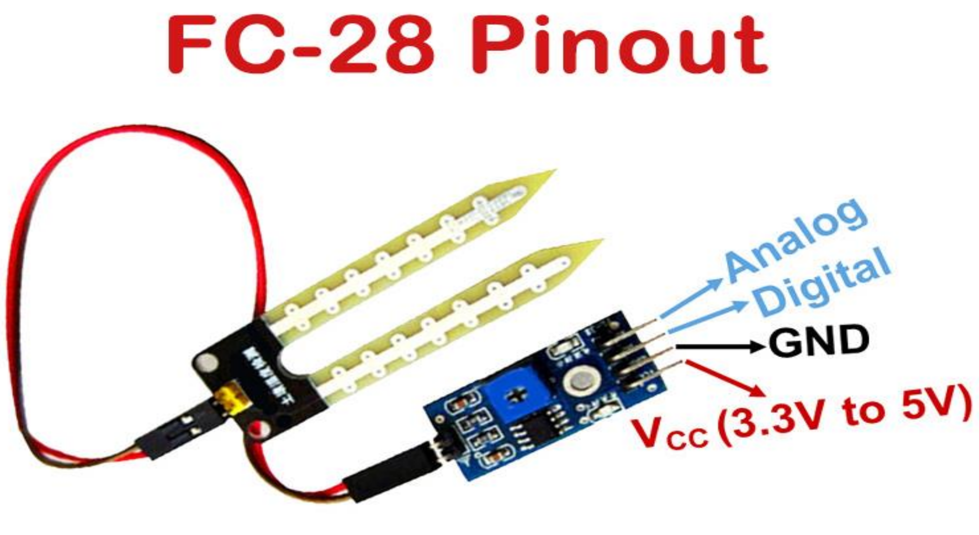


Fig 3.1.6: Soil Moisture Sensor Module

FC-28 moisture sensor probes are coated with an immersion gold that protects the Nickel probes from oxidation. These two probes are used to pass the current through the soil and then the sensor reads the resistance to get the moisture values. The sensing radius of the module depends upon the design complexity of the module itself, A satellite-based passive microwave sensor will cover a very wide area of ground, while cheap Chinese hobby sensors can cover only 20-30 cm of land. The module is compatible with a wide array of microcontrollers and allows enhanced interface ability over a wide range of platforms.

3.1.6.1 Features of Soil Moisture Sensor

1. Digital Output Threshold Adjust Potentiometer
2. Power and Digital Output Indicator LEDs
3. Analog and Digital outputs
4. Mounting hole for easy installation
5. Easy to use with Microcontrollers or even with normal Digital/Analog IC

6. Small, cheap, and easily available
7. ESD protection
8. Undervoltage Protection

3.1.6.2 Pin Description of Soil Moisture Sensor

The FC-28 soil moisture sensor has three pins for easy connection to other devices. The VCC pin is for power, connecting to the positive power supply. The GND pin is the ground connection, linking to the negative power supply. The SIG pin outputs an analog signal representing soil moisture levels, making it compatible with microcontrollers like Arduino. This sensor helps monitor and control soil moisture, ensuring optimal conditions for plant growth in a straightforward and accessible way.

Pin Number	Pin Name	Function
1	Vcc	he Vcc pin powers the module, typically with +5V
2	Gnd	Power Supply Ground
3	A0	Digital Out Pin for Digital Output
4	D0	Analog Out Pin for Analog Output

Table 3.1.6.2 Pin Description of soil moisture sensor

3.1.7 Water Pump 5v

Micro dc 3-6v micro submersible pump mini water pump for fountain garden mini water circulation system diy project dc 3v to 6v submersible pump micro mini submersible water pump 3v to 6vdc water pump for diy dc pump for hobby kit mini submersible pump motor this is a low cost, small size submersible pump motor which can be operated from a 2.5 ~ 6V power supply. It can take up to 120 liters per hour with very low current consumption of 220ma. Just connect tube pipe to the motor outlet, submerge it in water and power it. Make sure that the water level is always higher than the motor. The dry run may damage the motor due to heating and it will also produce noise.

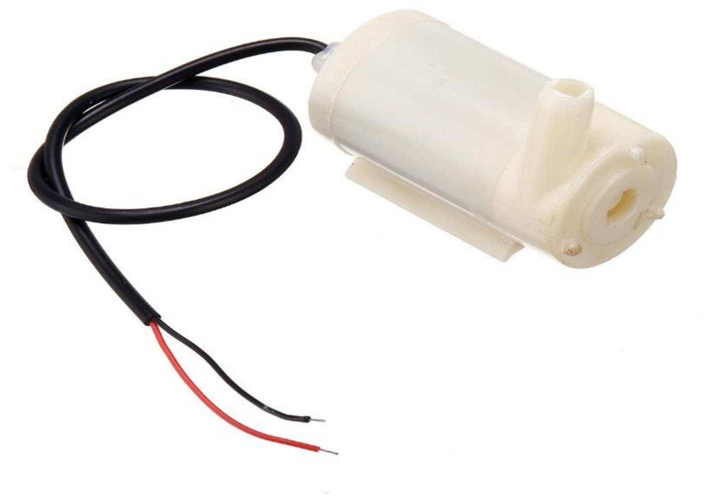


Fig 3.1.7 : 5v Water Pump

3.1.7.1 Features of 5v Water Pump

1. Voltage: 2.5-6V
2. Maximum lift: 40-110cm / 15.75"-43.4"
3. Flow rate: 80-120L/H
4. Outside diameter: 7.5mm / 0.3"
5. Inside diameter: 5mm / 0.2"
6. Diameter: Approx. 24mm / 0.95"
7. Length: Approx. 45mm / 1.8"
8. Height: Approx. 30mm / 1.2"
9. Material: Engineering plastic
10. Driving mode: DC design, magnetic driving

3.2 NODE MCU ESP32 MICRO-CONTROLLER

The ESP32 module is a highly versatile microcontroller developed by Espressif Systems. It boasts a dual-core Xtensa LX6 processor, providing enhanced processing capabilities for a variety of applications. Equipped with built-in Wi-Fi and Bluetooth support, the ESP32 is ideal for IoT projects, allowing seamless wireless communication. Its extensive set of GPIO pins, including digital and analog options, makes it adaptable for interfacing with sensors and peripherals. Security features, like hardware-accelerated encryption, enhance data protection in IoT applications. With compatibility for programming languages such as Arduino, Micro Python, and the Espressif IoT Development Framework, the ESP32 caters to a broad user base, from hobbyists to professionals.

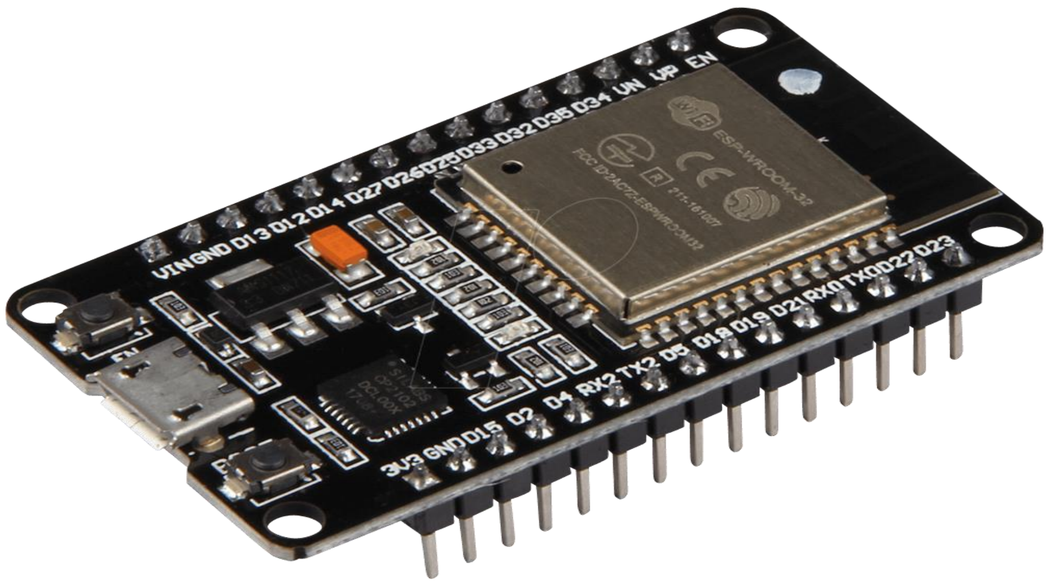


Fig 3.2 (a) ESP32 MICRO CONTROLLER MODULE

In addition to its technical prowess, the ESP32's popularity stems from its compact size, low power consumption, and user-friendly programmability. Its ability to function as a standalone microcontroller or connect to other devices via Wi-Fi and Bluetooth makes it a go-to choice for projects ranging from home automation to industrial applications. The ESP32's robust capabilities, ease of use, and strong community support have solidified its position as a key player in the ever-expanding landscape of microcontrollers and IoT development.

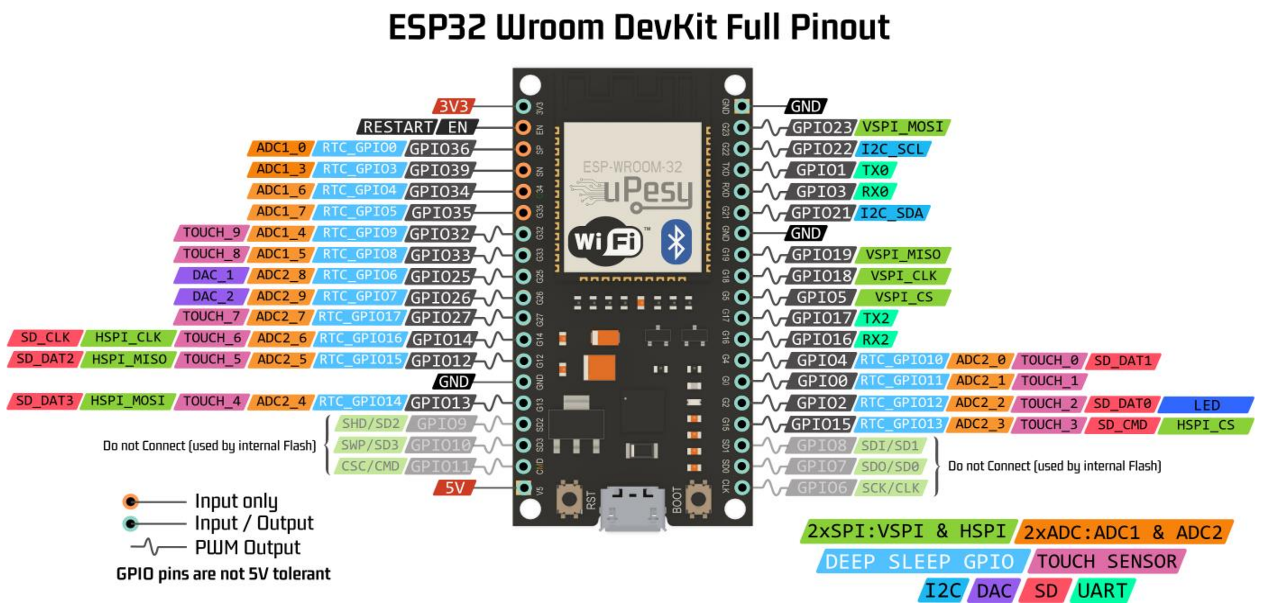


Fig 3.2 (b) PIN OUT OF ESP32 MICRO CONTROLLER MODULE

3.2.1 Applications

The ESP32 module finds applications in diverse fields, serving as the brain of smart home devices like thermostats and lighting systems. It powers wearable gadgets, enabling fitness trackers and smartwatches to connect seamlessly to other devices. In industrial settings, the ESP32 facilitates automation by controlling machinery and collecting data for monitoring. In agriculture, it aids in creating smart irrigation systems, while its use in educational projects helps students learn about electronics and programming. With its compact size and wireless capabilities, the ESP32 is a versatile choice for projects ranging from simple DIY setups to complex IoT applications.

3.3 Features of Micro controller

1. Dual-Core Processor: Dual-core Xtensa LX6 processor for improved processing capabilities.
2. Wireless Connectivity: Built-in Wi-Fi and Bluetooth
3. Peripheral Interfaces: Rich set of digital and analog input/output pins.

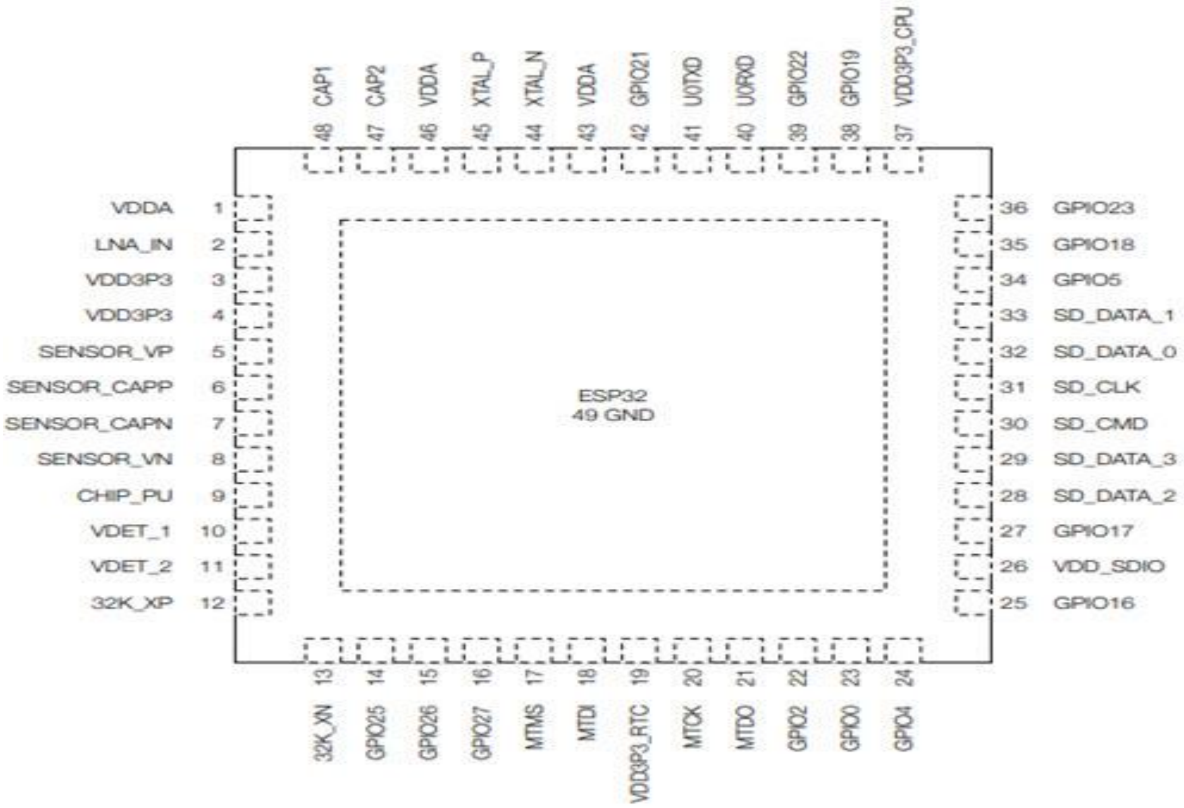
UART, SPI, I2C, PWM

4. 30 × programmable GPIOs
5. Memory: 520 KiB RAM, 448 KiB ROM
6. Ultra-low power (ULP) co-processor
7. IEEE 802.11 standard security features all supported, including WPA, WPA2, WPA3 (depending on version) [5] and WLAN Authentication and Privacy Infrastructure (WAPI)
8. Power management: Individual power domain for RTC
9. I/O pins: 28
10. Timers: Two 32 bit / Four 16-bit timers
11. PWM: 16 Channels
12. UART: Yes
13. 2 x I^{2}s interfaces

The ESP32 stands out with its dual-core processing power, enabling efficient multitasking and improved overall performance. Its built-in Wi-Fi and Bluetooth capabilities make it a go-to choice for seamless connectivity in Internet of Things (IoT) projects, allowing devices to communicate effortlessly. The microcontroller offers extensive peripheral support, featuring digital and analog pins, along with interfaces like UART, SPI, I2C, and PWM, providing versatility for connecting various sensors and peripherals. Security is a priority, with hardware-accelerated encryption ensuring the safe transmission of data, a critical feature for protecting sensitive information in IoT applications. The ESP32's ease of programmability, supporting popular languages like Arduino and Micro Python, caters to a diverse user base, making it accessible for both beginners and experienced developers. Moreover, its compact size and low power consumption make the ESP32 an ideal choice for a wide range of applications, from wearables to energy-efficient IoT devices.

3.4 PIN DIAGRAM OF Xtensa LX6 processor

The Xtensa LX6 processor in the ESP32 is a dual-core engine designed for efficient and powerful performance. With two processing units, it enables the ESP32 to handle multiple tasks simultaneously, enhancing overall responsiveness. The processor's architecture is optimized for embedded systems and Internet of Things (IoT) applications, providing a balance between performance and energy efficiency. Its dual-core design allows for effective multitasking, making the ESP32 well-suited for a variety of computing needs. The Xtensa LX6 contributes to the ESP32's capability to execute complex instructions, making it a reliable choice for a wide range of projects, from smart home devices to industrial automation.



DESCRIPTION OF EACH PIN OF ESP32 MICRO-CONTROLLER MODULE

Pin number	**Description**	**Function**
1	VDDA	Analog power supply (2.3 V ~ 3.6 V)
2	LNA_IN	RF input and output
3	VDD3P3	Analog power supply (2.3 V ~ 3.6 V)
4	VDD3P3	Analog power supply (2.3 V ~ 3.6 V)
5	SENSOR_VP	GPIO36, ADC1_CH0, RTC_GPIO0
6	SENSOR_CAPP	GPIO37, ADC1_CH1, RTC_GPIO1
7	SENSOR_CAPN	GPIO38, ADC1_CH2, RTC_GPIO2
8	SENSOR_CN	GPIO39, ADC1_CH3, RTC_GPIO3
9	CHIP_PU	**High**: On; enables the chip **Low**: Off; the chip shuts down
10	VDET_1	GPIO34, ADC1_CH6, RTC_GPIO4
11	VDET_2	GPIO35, ADC1_CH7, RTC_GPIO5
12	32K_XP	GPIO32, ADC1_CH4, RTC_GPIO9, TOUCH9, 32K_XP (32.768 kHz crystal oscillator input)
13	32K_XN	GPIO33, ADC1_CH5, RTC_GPIO8, TOUCH8, 32K_XN (32.768 kHz crystal oscillator output)
14	GPIO25	GPIO25, ADC2_CH8, RTC_GPIO6, DAC_1, EMAC_RXD0

15	GPIO26	GPIO26, ADC2_CH9, RTC_GPIO7, DAC_2, EMAC_RXD1
16	GPIO27	GPIO27, ADC2_CH7, RTC_GPIO17, TOUCH7, EMAC_RX_DV
17	MTMS	GPIO14, ADC2_CH6, RTC_GPIO16, TOUCH6, EMAC_TXD2, HSPICLK, HS2_CLK, SD_CLK
18	MTDI	GPIO12, ADC2_CH5, RTC_GPIO15, TOUCH5, EMAC_TXD3, HSPIQ, HS2_DATA2, SD_DATA2
19	VDD3P3_RTC	Input power supply for RTC IO (2.3 V ~ 3.6 V)
20	MTCK	GPIO13, ADC2_CH4, RTC_GPIO14, TOUCH4, EMAC_RX_ER, HSPID, HS2_DATA3, SD_DATA3
21	MTDO	GPIO15, ADC2_CH3, RTC_GPIO13, TOUCH3, EMAC_RXD3, HSPICS0, HS2_CMD, SD_CMD
22	GPIO2	ADC2_CH2, RTC_GPIO12, TOUCH2, HSPIWP, HS2_DATA0, SD_DATA0
23	GPIO0	ADC2_CH1, RTC_GPIO11, TOUCH1, EMAC_TX_CLK, CLK_OUT1,
24	GPIO4	GPIO4, ADC2_CH0, RTC_GPIO10, TOUCH0, EMAC_TX_ER, HSPIHD, HS2_DATA1, SD_DATA1
25	GPIO16	GPIO16, HS1_DATA4, U2RXD,
26	VDD_SDIO	Output power supply: 1.8 V or the same voltage as VDD3P3_RTC
27	GPIO17	GPIO17, HS1_DATA5, U2TXD, EMAC_CLK_OUT_180
28	SD_DATA_2	GPIO9, HS1_DATA2, U1RXD, SD_DATA2, SPIHD

29	SD_DATA_3	GPIO10, HS1_DATA3, U1TXD, SD_DATA3, SPIWP
30	SD_CMD	GPIO11, HS1_CMD, U1RTS, SD_CMD, SPICS0
31	SD_CLK	GPIO6, HS1_CLK, U1CTS, SD_CLK, SPICLK
32	SD_DATA_0	GPIO7, HS1_DATA0, U2RTS, SD_DATA0, SPIQ
33	SD_DATA_1	GPIO8, HS1_DATA1, U2CTS, SD_DATA1, SPID
34	GPIO5	GPIO5, HS1_DATA6, VSPICS0, EMAC_RX_CLK
35	GPIO18	GPIO18, HS1_DATA7, VSPICLK
36	GPIO23	GPIO23, HS1_STROBE, VSPID
37	VDD3P3_CPU	Input power supply for CPU IO (1.8 V ~ 3.6 V)
38	GPIO19	GPIO19, U0CTS, VSPIQ
39	GPIO22	GPIO22, U0RTS, VSPIWP, EMAC_TXD1
40	U0RXD	GPIO3, U0RXD, CLK_OUT2
41	U0TXD	GPIO1, U0TXD, CLK_OUT3, EMAC_RXD2
42	GPIO21	GPIO21, VSPIHD, EMAC_TX_EN
43	VDDA	Analog power supply (2.3 V ~ 3.6 V)
44	XTAL_N	External crystal output
45	XTAL_P	External crystal input

46	VDDA	Analog power supply (2.3 V ~ 3.6 V)
47	CAP2	Connects to a 3.3 nF (10%) capacitor and 20 k Ω resistor in parallel to CAP1
48	CAP1	Connects to a 10 nF series capacitor to ground

Table 3.5: Pin Description of Xtensa Lx6 Processor

3.5 BASIC TERMINOLOGIES IN ESP32

3.5.1 Analog to Digital Converter (ADC)

1. **ADC Channels:** The ESP32 has multiple ADC channels, typically labeled as ADC1_CH0, ADC1_CH1, ..., ADC2_CH0, ADC2_CH1, etc. Each channel corresponds to a specific pin on the ESP32 that can be used for analog input.
2. **Voltage Range:** The ADC in the ESP32 typically operates in the range of 0 to 3.3 volts. Make sure that the analog signal you're measuring falls within this range to avoid damaging the ADC.
3. **Resolution:** The ADC in the ESP32 has a configurable resolution, usually ranging from 9 to 12 bits. Higher resolution provides more precise measurements but requires more processing time.
4. **Reference Voltage:** The ESP32 allows you to set the reference voltage for the ADC. The default reference voltage is usually the supply voltage (VDD), but you can also use an external reference voltage for more accurate measurements.
5. **Functionality:** The ESP32 ADC can be used in both single-ended and differential modes. In single-ended mode, the voltage is measured with respect to ground, while in differential mode, the voltage difference between two channels is measured.

6. **Reading ADC Values:** To read analog values using the ADC in the ESP32, you typically use functions provided by the ESP32 SDK or the Arduino IDE. In Arduino, you may use functions like **analogRead( )** to obtain the ADC values.

3.5.2 Pulse Width Modulation (PWM)

1. **PWM Channels:** The ESP32 provides multiple PWM channels, each associated with a specific GPIO pin. The number of PWM channels available depends on the specific ESP32 module or development board you are using.
2. **Frequency and Duty Cycle:** PWM allows you to control the frequency and duty cycle of the output signal. The frequency determines how fast the PWM signal oscillates, while the duty cycle represents the percentage of time the signal is high compared to the total period. Higher duty cycles correspond to higher average voltages.
3. **Pins with PWM Support:** Not all GPIO pins on the ESP32 support PWM. You need to refer to the documentation or pinout diagram of your specific ESP32 module to identify which pins are PWM-enabled.
4. **Using PWM in Arduino IDE:** If you're using the Arduino IDE to program the ESP32, you can use the **analogWrite()** function to generate PWM signals. The function takes a pin number and a duty cycle value (ranging from 0 to 255).

CHAPTER – 4

SOFTWARE SYSTEM REQURIMENTS

4.1 Language Reference

The ESP32, a tiny computer, can be programmed using two main languages: C and C++. If you're using the official Espressif IoT Development Framework (ESP-IDF), you'll mainly work with C. There's a helpful documentation on the Espressif website that explains how to use different functions and features for ESP32. On the other hand, if you're using the Arduino framework, you'll write code in a simpler version of C++ that is more beginner-friendly. The Arduino framework hides some of the complicated details, making it easier for people who are just starting. You can find information about ESP32 functions and libraries in the Arduino reference. Whether you choose ESP-IDF or Arduino depends on your project needs and how comfortable you are with each approach - whether you want more control (ESP-IDF) or an easier way to get started (Arduino).

4.2 SOFTWARE REQUIREMENTS

4.2.1 Embedded C

Embedded C programming for the ESP32 involves using the C programming language to develop software for the microcontroller. The ESP32 is a versatile microcontroller with integrated Wi-Fi and Bluetooth capabilities, and it's commonly programmed using the Espressif IoT Development Framework (ESP-IDF). Embedded C, in the context of ESP32, involves writing code that directly interacts with the hardware features of the microcontroller, such as GPIO pins, peripherals (like ADC and PWM), and communication interfaces (Wi-Fi and Bluetooth). The ESP-IDF provides a set of APIs, libraries, and documentation that allow developers to harness the full potential of the ESP32. The code written in Embedded C for ESP32 is typically more low-level and hardware-centric compared to using higher-level frameworks like Arduino, providing greater control and optimization possibilities. Understanding the ESP-IDF documentation and APIs is crucial for efficiently utilizing the ESP32's features in Embedded C programming.

4.2.2 Different between C and Embedded C

C and Embedded C are essentially the same language, but their usage contexts distinguish them. C is a versatile, general-purpose programming language employed in diverse applications, from desktop software to server development. On the other hand, Embedded C is a subset of C crafted specifically for programming embedded systems, such as microcontrollers and IoT devices, which operate under resource constraints. Embedded C focuses on efficiency in managing limited memory and system resources, often providing specialized libraries and features for low-level programming and hardware interfacing. While C assumes ample resources, Embedded C is tailored to optimize code for size and performance in the embedded system environment, requiring specific tools and compilers provided by microcontroller manufacturers for development.

4.2.3 HTML

In your project, HTML (Hypertext Markup Language) plays a crucial role in creating the structure and layout of the web interface. HTML is like the blueprint of a webpage, defining elements such as titles, paragraphs, and links. In simple terms, it helps organize and present information on the monitoring system's display. For example, it specifies where to show temperature, humidity, and other sensor data. Additionally, by embedding HTML code in your Arduino sketch, you're able to dynamically update and showcase real-time information, making it accessible and visually appealing for users monitoring the environmental conditions through a web browser.

4.2.4 CSS

In your project, CSS (Cascading Style Sheets) is employed to enhance the visual presentation and appearance of the web interface created with HTML. CSS acts like a stylist, determining the colors, fonts, and layout of the displayed elements. It allows you to make the monitoring system interface not only informative but also visually engaging and user-friendly. For instance, CSS is used to define the background colors, text styles, and animations, giving a polished look to the temperature, humidity, and other sensor readings. By incorporating CSS into your project, you ensure a pleasant and well-designed user experience, making the monitoring system visually appealing and easy to navigate.

4.2.5 JAVA SCRIPT

In your project, JavaScript is utilized to dynamically update and refresh sensor data on the web interface without requiring the user to manually reload the page. Acting as the interactive engine, JavaScript constantly communicates with the server, fetching and displaying real-time information such as temperature, humidity, and other sensor readings. This enhances the user experience by providing timely and automatic updates, ensuring that the displayed data is always current. JavaScript's role is like a behind-the-scenes coordinator, making the monitoring system responsive and efficient in delivering live sensor information to users as it becomes available.

4.3 ARDUINO IDE

Fig 4.3 : Arduino IDE Logo

The Arduino IDE (Integrated Development Environment) is a user-friendly software tool that helps people create and program projects using Arduino microcontrollers. It serves as a virtual workspace where you write, edit, and upload code to control electronic components. Think of it like a digital workshop where you can easily design, experiment, and bring your ideas to life, even if you're not an expert in programming. The IDE simplifies the process of coding for Arduino, making it accessible for beginners and experienced makers alike, providing a platform to turn creative concepts into functioning electronic projects.

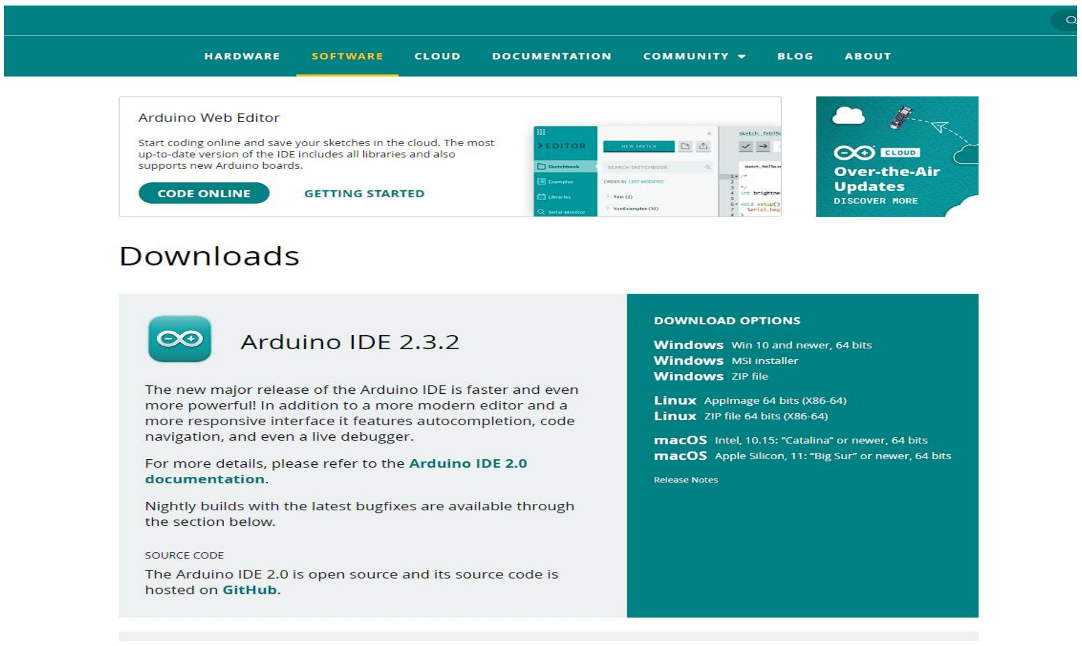
4.3.1 UPLOADING OF SOFTWARE TO ESP32 MICRO-CONTROLLER

Uploading software to an ESP32 board can be accomplished through the Arduino IDE or the Espressif IoT Development Framework (ESP-IDF). In the Arduino IDE, after installing the software and ESP32 board support, select the appropriate board, connect the ESP32 via USB, choose the correct COM port, and click "Upload" to transfer your code. Alternatively, when using the ESP-IDF, install and configure the framework, set up environment variables, navigate to your project directory, configure project settings, and finally build and flash the code to the ESP32 using the command line. Monitoring the serial output can be done optionally to view debug information. Specific details may vary, so consulting the respective documentation for the chosen development environment is recommended for precise instructions.

4.4 Steps to Upload Code to ESP32 using Arduino IDE

4.4.1 Install Arduino IDE

- Visit the official Arduino website at https://www.arduino.cc/en/software. Click on the "Software" tab, and then select the appropriate version for your operating system (Windows, macOS, or Linux).



Arduino Web Editor

Start coding online and save your sketches in the cloud. The most up-to-date version of the IDE includes all libraries and also supports new Arduino boards.

Downloads

Arduino IDE 2.3.2

The new major release of the Arduino IDE is faster and even more powerful! In addition to a more modern editor and a more responsive interface it features autocompletion, code navigation, and even a live debugger.

For more details, please refer to the **Arduino IDE 2.0 documentation**.

Nightly builds with the latest bugfixes are available through the section below.

SOURCE CODE

The Arduino IDE 2.0 is open source and its source code is hosted on **GitHub**.

DOWNLOAD OPTIONS

Windows Win 10 and newer, 64 bits
Windows MSI installer
Windows ZIP file

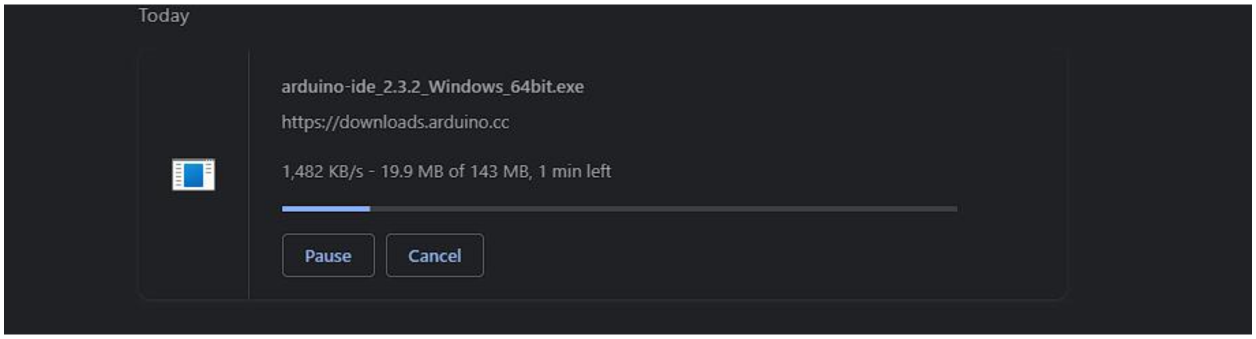
Linux AppImage 64 bits (X86-64)
Linux ZIP file 64 bits (X86-64)

macOS Intel, 10.15: "Catalina" or newer, 64 bits
macOS Apple Silicon, 11: "Big Sur" or newer, 64 bits

Release Notes

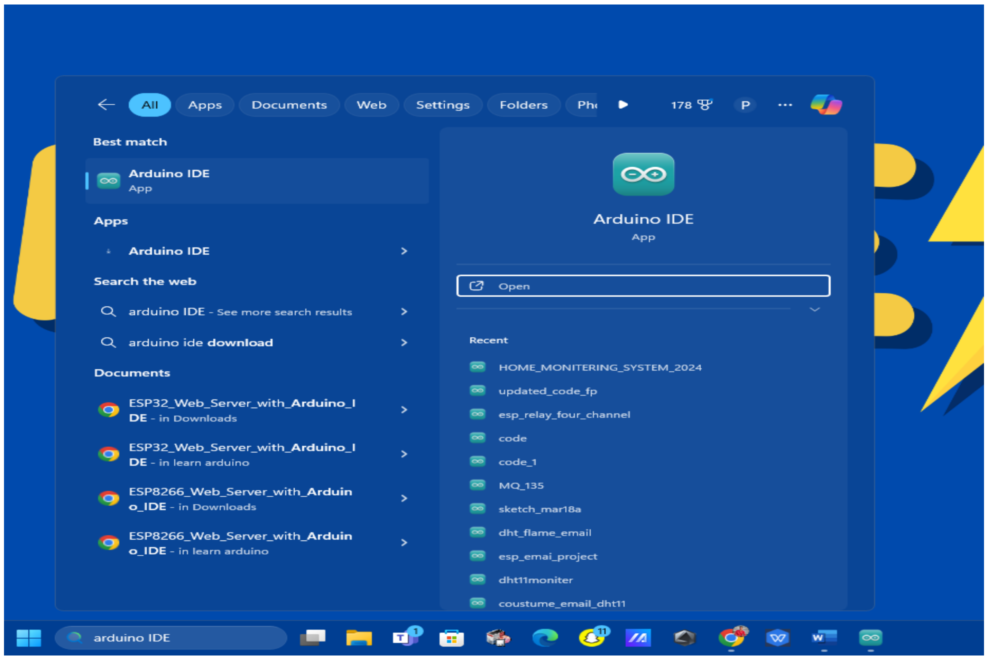
- Click on Windows win10 and newer,64bit download

- And click just download and it will download, press ctrl+j for download conformation


Today
arduino-ide_2.3.2_Windows_64bit.exe
https://downloads.arduino.cc
1,482 KB/s - 19.9 MB of 143 MB, 1 min left
Pause
Cancel

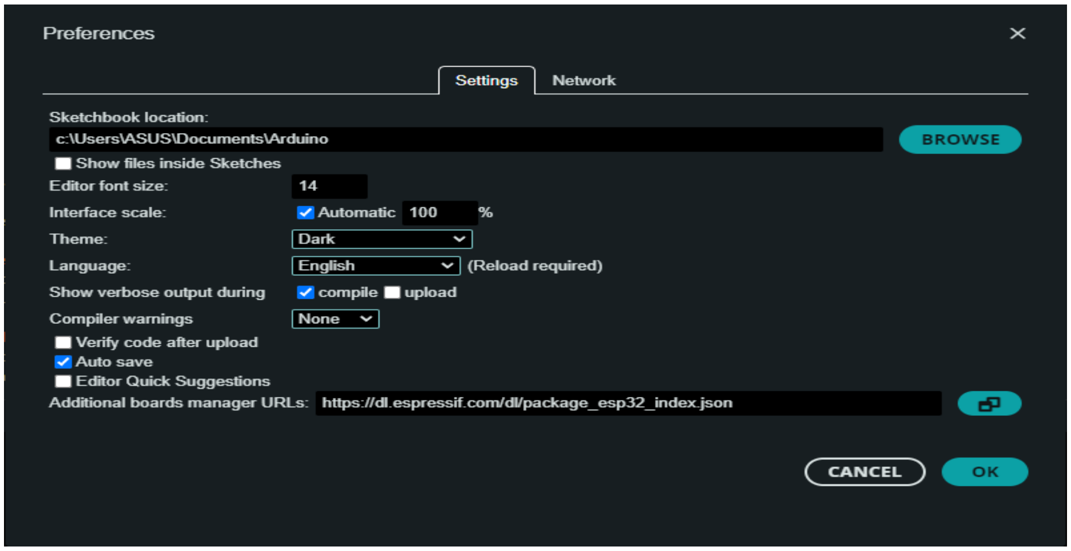
4.4.2 <u>Run Installer</u>

Once the download is complete, run the installer file that you obtained in the previous step.



4.4.3 Install ESP32 Board Support

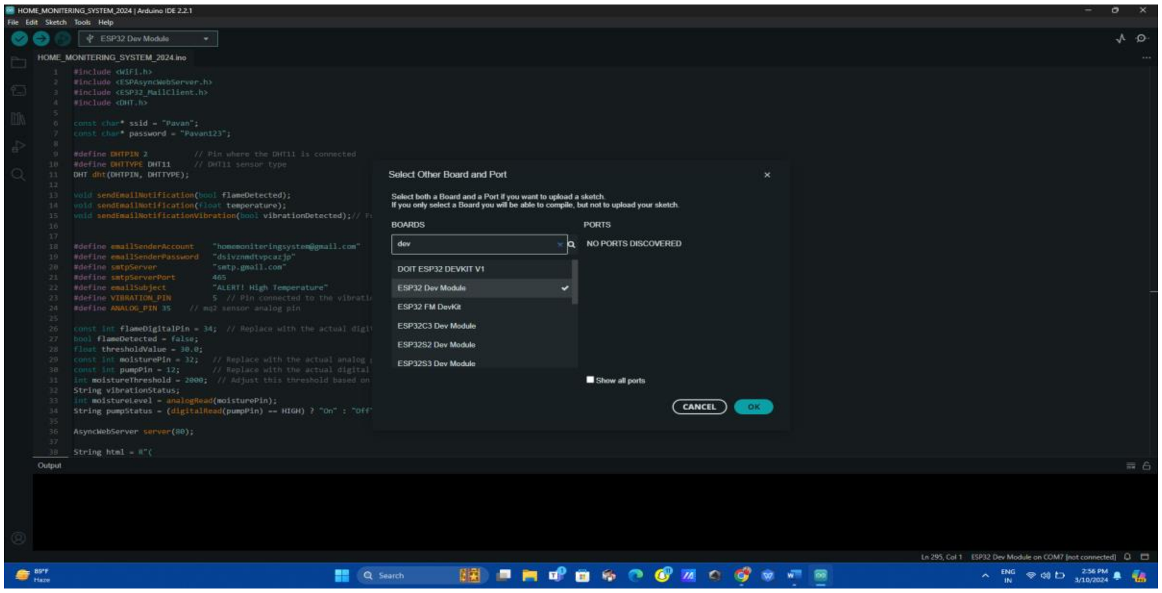
- Open the Arduino IDE, go to "File" > "Preferences," and add the ESP32 board support URL to the Additional Boards Manager URLs. The URL is: https://dl.espressif.com/dl/package_esp32_index.json
- Then, go to "Tools" > "Board" > "Boards Manager," search for "esp32," and install the ESP32 board support.



Sketchbook location:
c:\Users\ASUS\Documents\Arduino
☐ Show files inside Sketches
Editor font size: 14
Interface scale: ✓ Automatic 100 %
Theme: Dark
Language: English (Reload required)
Show verbose output during ✓ compile ☐ upload
Compiler warnings None
☐ Verify code after upload
✓ Auto save
☐ Editor Quick Suggestions
Additional boards manager URLs: https://dl.espressif.com/dl/package_esp32_index.json

4.4.4 Select ESP32 Board

- In the Arduino IDE, go to "Tools" > "Board" and select your specific ESP32 board from the list. Choose the appropriate variant and upload speed.
- After following these steps, you should have a fully installed Arduino IDE on your computer, ready to be used for programming Arduino boards, including the ESP32.



```
#include <WiFi.h>
#include <ESPAsyncWebServer.h>
#include <ESP32_MailClient.h>
#include <DHT.h>

const char* ssid = "Pavan";
const char* password = "Pavan123";

#define DHTPIN 2          // Pin where the DHT11 is connected
#define DHTTYPE DHT11     // DHT11 sensor type
DHT dht(DHTPIN, DHTTYPE);

void sendEmailNotification(bool flameDetected);
void sendEmailNotification(float temperature);
void sendEmailNotificationVibration(bool vibrationDetected);// F


#define emailSenderAccount    "homemoniteringsystem@gmail.com"
#define emailSenderPassword   "dsivznmdtvpcazjp"
#define smtpServer            "smtp.gmail.com"
#define smtpServerPort        465
#define emailSubject          "ALERT! High Temperature"
#define VIBRATION_PIN         5  // Pin connected to the vibrati
#define ANALOG_PIN 35    // mq2 sensor analog pin

const int flameDigitalPin = 34;  // Replace with the actual digit
bool flameDetected = false;
float thresholdValue = 30.0;
const int moisturePin = 32;   // Replace with the actual analog
const int pumpPin = 12;       // Replace with the actual digital
int moistureThreshold = 2000;  // Adjust this threshold based on
String vibrationStatus;
int moistureLevel = analogRead(moisturePin);
String pumpStatus = (digitalRead(pumpPin) == HIGH) ? "On" : "Off"

AsyncWebServer server(80);

String html = R"(
```

Select Other Board and Port

Select both a Board and a Port if you want to upload a sketch.
If you only select a Board you will be able to compile, but not to upload your sketch.

BOARDS: dev

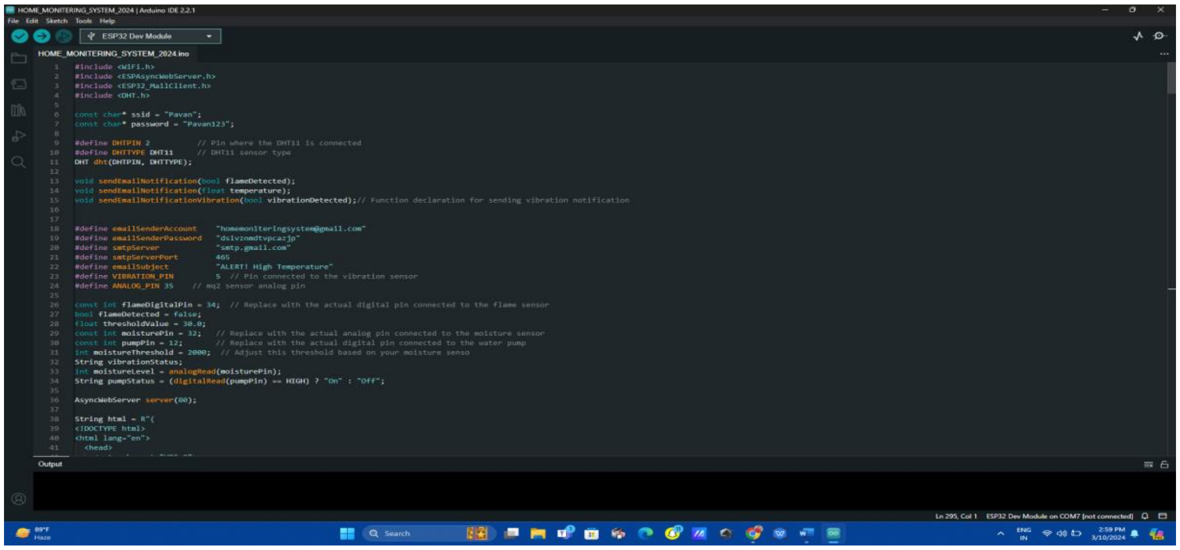
- DOIT ESP32 DEVKIT V1
- ESP32 Dev Module ✓
- ESP32 FM DevKit
- ESP32C3 Dev Module
- ESP32S2 Dev Module
- ESP32S3 Dev Module

PORTS: NO PORTS DISCOVERED

HOME_MONITERING_SYSTEM_2024 | Arduino IDE 2.2.1
File Edit Sketch Tools Help
ESP32 Dev Module
HOME_MONITERING_SYSTEM_2024.ino
Show all ports
CANCEL
OK
Output
Ln 295, Col 1 ESP32 Dev Module on COM7 [not connected]

4.4.5 WRITE CODE

Write your Arduino sketch or open an existing one. Ensure that your code is compatible with the ESP32 and contains the necessary configurations.

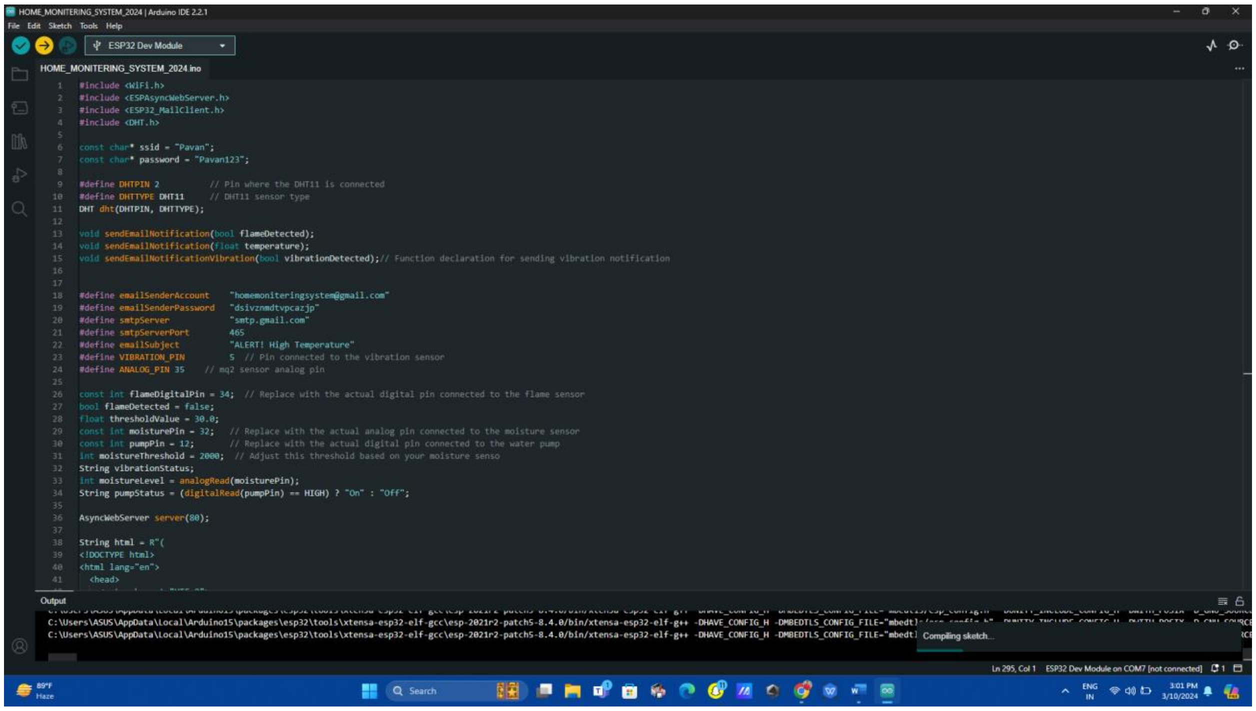


```
#include <WiFi.h>
#include <ESPAsyncWebServer.h>
#include <ESP32_MailClient.h>
#include <DHT.h>

const char* ssid = "Pavan";
const char* password = "Pavan123";

#define DHTPIN 2          // Pin where the DHT11 is connected
#define DHTTYPE DHT11     // DHT11 sensor type
DHT dht(DHTPIN, DHTTYPE);

void sendEmailNotification(bool flameDetected);
void sendEmailNotification(float temperature);
void sendEmailNotificationVibration(bool vibrationDetected);// Function declaration for sending vibration notification


#define emailSenderAccount    "homemoniteringsystem@gmail.com"
#define emailSenderPassword   "dsivznmdtvpcazjp"
#define smtpServer            "smtp.gmail.com"
#define smtpServerPort        465
#define emailSubject          "ALERT! High Temperature"
#define VIBRATION_PIN         5  // Pin connected to the vibration sensor
#define ANALOG_PIN 35    // mq2 sensor analog pin

const int flameDigitalPin = 34;  // Replace with the actual digital pin connected to the flame sensor
bool flameDetected = false;
float thresholdValue = 30.0;
const int moisturePin = 32;   // Replace with the actual analog pin connected to the moisture sensor
const int pumpPin = 12;       // Replace with the actual digital pin connected to the water pump
int moistureThreshold = 2000;  // Adjust this threshold based on your moisture senso
String vibrationStatus;
int moistureLevel = analogRead(moisturePin);
String pumpStatus = (digitalRead(pumpPin) == HIGH) ? "On" : "Off";

AsyncWebServer server(80);

String html = R"(
<!DOCTYPE html>
<html lang="en">
  <head>
```

HOME_MONITERING_SYSTEM_2024 | Arduino IDE 2.2.1
File Edit Sketch Tools Help
ESP32 Dev Module
HOME_MONITERING_SYSTEM_2024.ino
Output
Ln 295, Col 1 ESP32 Dev Module on COM7 [not connected]

4.4.6 Upload Code

Click the "Upload" button (right arrow icon) in the Arduino IDE. This will compile your code and upload it to the ESP32. The status will be displayed in the bottom console, and any errors will be shown if there are issues during the upload process.


HOME_MONITERING_SYSTEM_2024 | Arduino IDE 2.2.1
File Edit Sketch Tools Help
ESP32 Dev Module
HOME_MONITERING_SYSTEM_2024.ino

```
#include <WiFi.h>
#include <ESPAsyncWebServer.h>
#include <ESP32_MailClient.h>
#include <DHT.h>

const char* ssid = "Pavan";
const char* password = "Pavan123";

#define DHTPIN 2          // Pin where the DHT11 is connected
#define DHTTYPE DHT11     // DHT11 sensor type
DHT dht(DHTPIN, DHTTYPE);

void sendEmailNotification(bool flameDetected);
void sendEmailNotification(float temperature);
void sendEmailNotificationVibration(bool vibrationDetected);// Function declaration for sending vibration notification


#define emailSenderAccount    "homemoniteringsystem@gmail.com"
#define emailSenderPassword   "dsivznmdtvpcazjp"
#define smtpServer            "smtp.gmail.com"
#define smtpServerPort        465
#define emailSubject          "ALERT! High Temperature"
#define VIBRATION_PIN         5  // Pin connected to the vibration sensor
#define ANALOG_PIN 35    // mq2 sensor analog pin

const int flameDigitalPin = 34;  // Replace with the actual digital pin connected to the flame sensor
bool flameDetected = false;
float thresholdValue = 30.0;
const int moisturePin = 32;   // Replace with the actual analog pin connected to the moisture sensor
const int pumpPin = 12;       // Replace with the actual digital pin connected to the water pump
int moistureThreshold = 2000;  // Adjust this threshold based on your moisture senso
String vibrationStatus;
int moistureLevel = analogRead(moisturePin);
String pumpStatus = (digitalRead(pumpPin) == HIGH) ? "On" : "Off";

AsyncWebServer server(80);

String html = R"(
<!DOCTYPE html>
<html lang="en">
  <head>
```

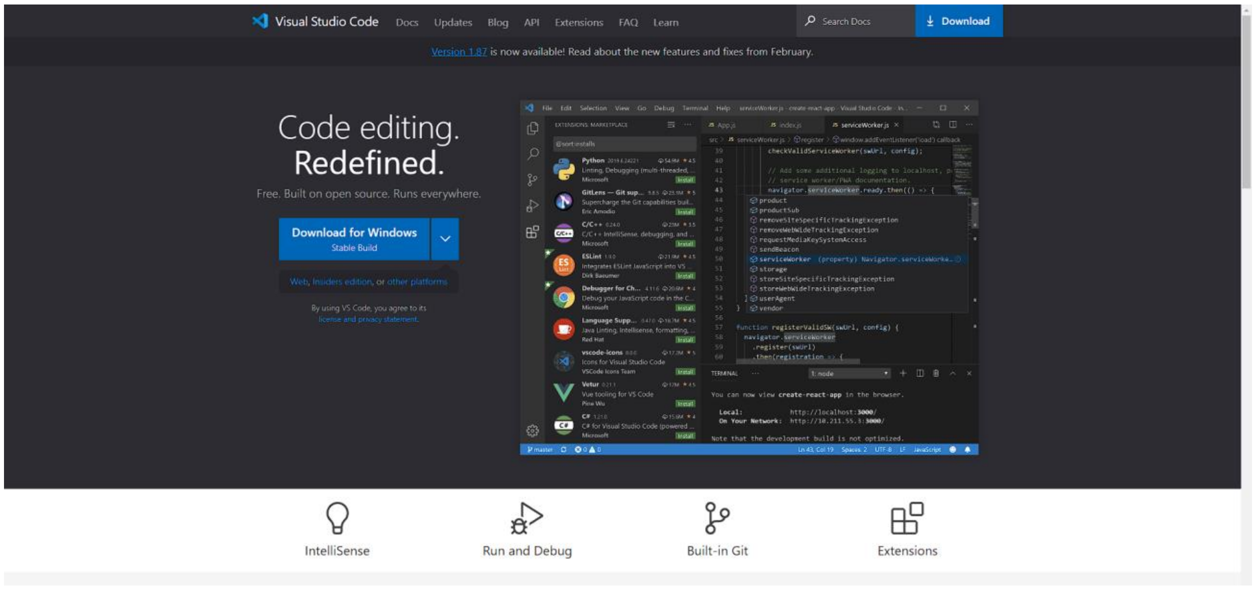
Output

```
C:\Users\ASUS\AppData\Local\Arduino15\packages\esp32\tools\xtensa-esp32-elf-gcc\esp-2021r2-patch5-8.4.0/bin/xtensa-esp32-elf-g++ -DHAVE_CONFIG_H -DMBEDTLS_CONFIG_FILE="mbedtl
C:\Users\ASUS\AppData\Local\Arduino15\packages\esp32\tools\xtensa-esp32-elf-gcc\esp-2021r2-patch5-8.4.0/bin/xtensa-esp32-elf-g++ -DHAVE_CONFIG_H -DMBEDTLS_CONFIG_FILE="mbedtl
```

4.5 USER INTERFACE USING HTML AND CSS

4.5.1 Install Visual Studio Code

Download and install Visual Studio Code from the official website: Visual Studio Code



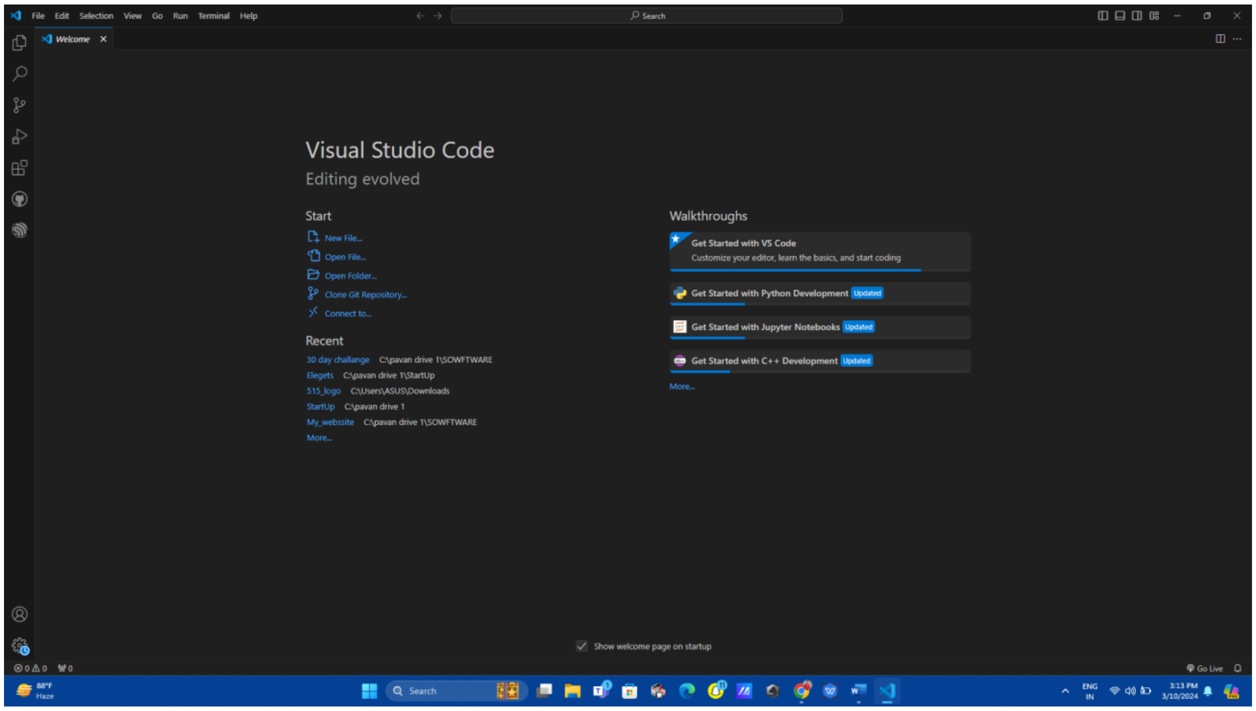
Version 1.87 is now available! Read about the new features and fixes from February.

Code editing.
Redefined.

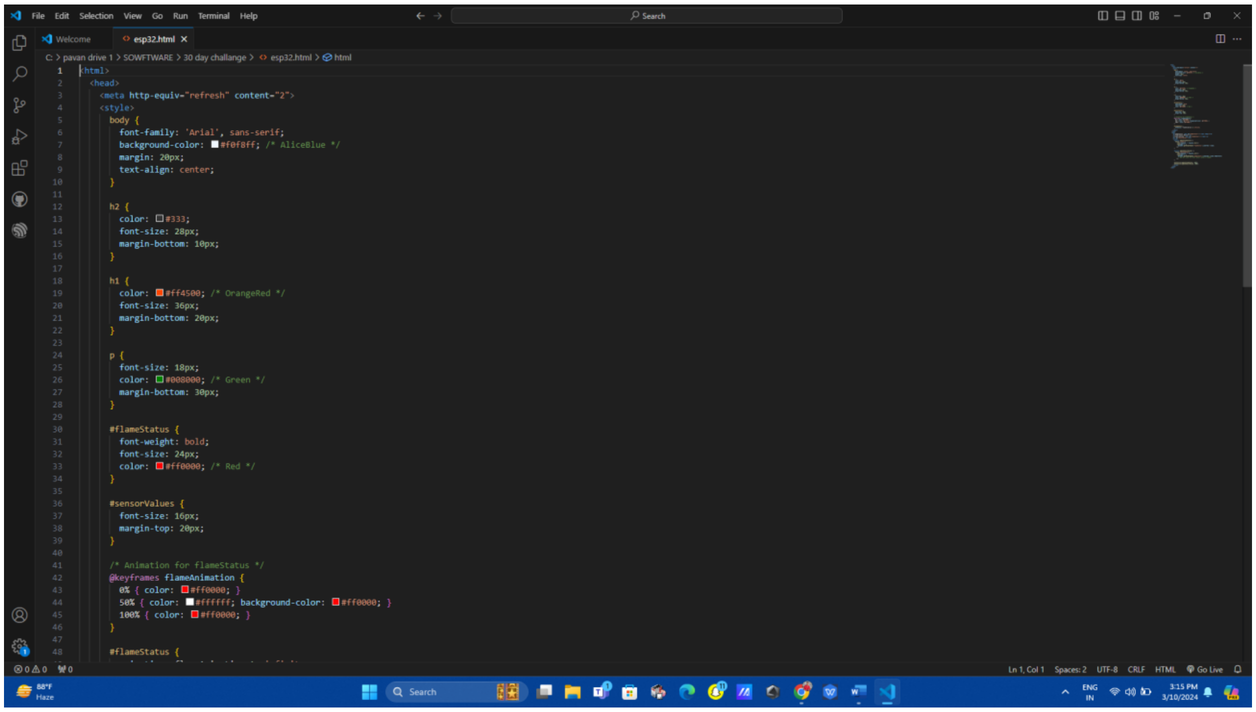
Free. Built on open source. Runs everywhere.

By using VS Code, you agree to its license and privacy statement.

Open VS Code and go to the Extensions view by clicking on the Extensions icon in the Activity Bar on the side of the window or press **Ctrl+Shift+X**.


Visual Studio Code
Editing evolved
Start
New File...
Open File...
Open Folder...
Clone Git Repository...
Connect to...
Recent
30 day challange C:\pavan drive 1\SOWFTWARE
Elegets C:\pavan drive 1\StartUp
515_logo C:\Users\ASUS\Downloads
StartUp C:\pavan drive 1
My_webssite C:\pavan drive 1\SOWFTWARE
More...
Walkthroughs
Get Started with VS Code
Customize your editor, learn the basics, and start coding
Get Started with Python Development Updated
Get Started with Jupyter Notebooks Updated
Get Started with C++ Development Updated
More...
Show welcome page on startup

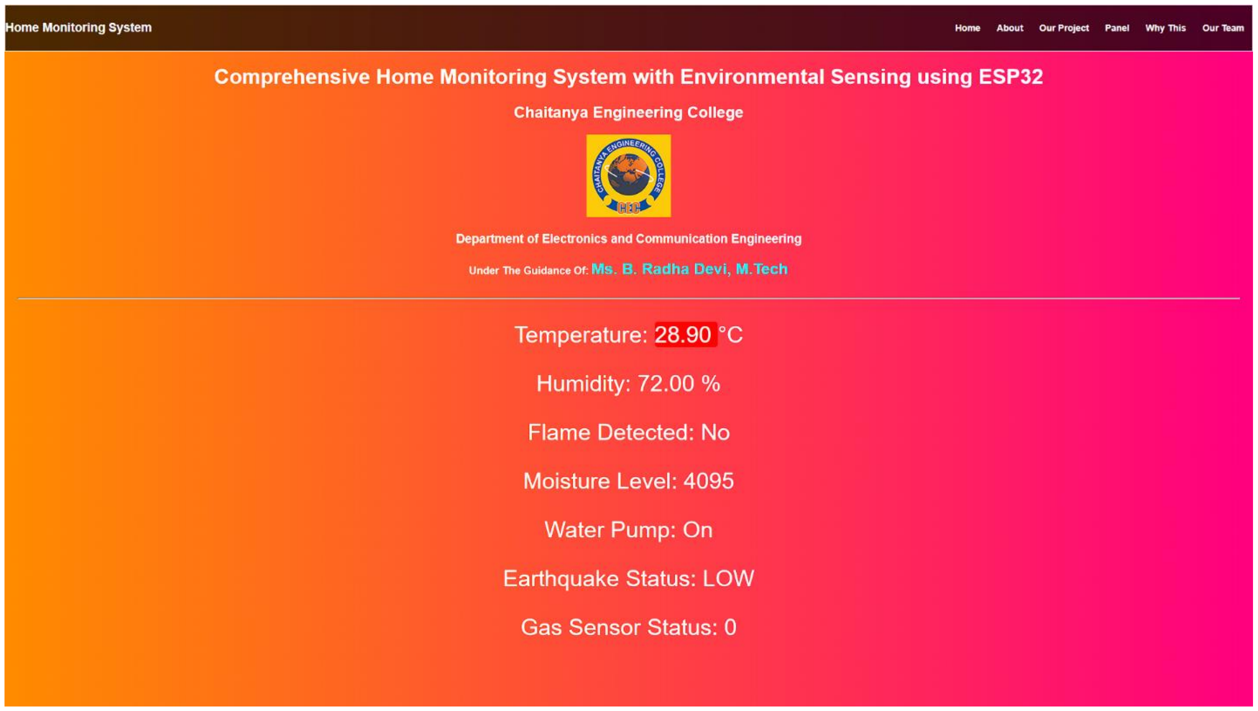
- Click on "New Project" and select your board (ESP32). Follow the prompts to set up a new project.
- Open your HTML and CSS files in the VS Code editor and start coding your user interface. You can use standard HTML and CSS syntax.



C: > pavan drive 1 > SOWFTWARE > 30 day challange > esp32.html > html

```
<html>
  <head>
    <meta http-equiv="refresh" content="2">
    <style>
      body {
        font-family: 'Arial', sans-serif;
        background-color: #f0f8ff; /* AliceBlue */
        margin: 20px;
        text-align: center;
      }

      h2 {
        color: #333;
        font-size: 28px;
        margin-bottom: 10px;
      }

      h1 {
        color: #ff4500; /* OrangeRed */
        font-size: 36px;
        margin-bottom: 20px;
      }

      p {
        font-size: 18px;
        color: #008000; /* Green */
        margin-bottom: 30px;
      }

      #flameStatus {
        font-weight: bold;
        font-size: 24px;
        color: #ff0000; /* Red */
      }

      #sensorValues {
        font-size: 16px;
        margin-top: 20px;
      }

      /* Animation for flameStatus */
      @keyframes flameAnimation {
        0% { color: #ff0000; }
        50% { color: #ffffff; background-color: #ff0000; }
        100% { color: #ff0000; }
      }

      #flameStatus {
```

- Check preview by clicking run / Alt+B


Home Monitoring System

Home About Our Project Panel Why This Our Team

Comprehensive Home Monitoring System with Environmental Sensing using ESP32

Chaitanya Engineering College

Department of Electronics and Communication Engineering

Under The Guidance Of: Ms. B. Radha Devi, M.Tech

Temperature: 28.90 °C

Humidity: 72.00 %

Flame Detected: No

Moisture Level: 4095

Water Pump: On

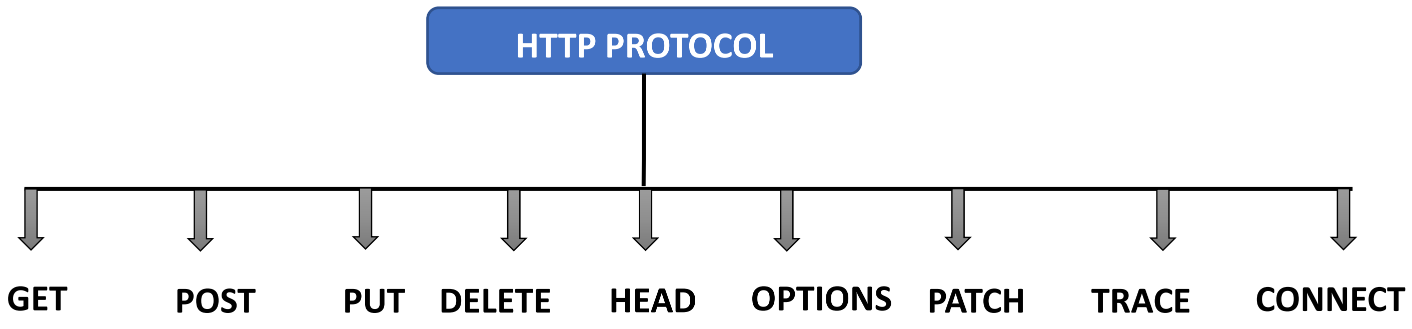
Earthquake Status: LOW

Gas Sensor Status: 0

- Upload code from **APPENDIX** to esp32 module and connect the sensors as per circuit and check the values

4.6 HTTP PROTOCOL

In your project, the HTTP (Hypertext Transfer Protocol) protocol serves as the communication method between the ESP32-based monitoring system and the web browser. It's like a set of rules that allows the devices to exchange information. When you access the monitoring system through a web browser, the browser uses HTTP to request data from the ESP32. The ESP32 then responds, providing the requested information, such as sensor readings and system status. It's similar to a conversation where one side asks questions (HTTP requests), and the other side provides answers (HTTP responses), enabling seamless and standardized communication between the monitoring system and the user's browser.



TYPES OF HTTP PROTOCOLS

4.6.1 Explanation of commonly used HTTP Protocol

1) GET - Requests data from a specified resource. It should have no side effects on the server.

2) POST - Submits data to be processed to a specified resource. It may lead to a change in state on the server.

3) PUT - Updates a resource or creates a new resource if it does not exist at the specified URI.

4) DELETE - Deletes the specified resource.

5) HEAD - Requests the headers of the specified resource without the actual data. It's often used to check if a resource has changed.

6) OPTIONS - Requests information about the communication options available for the target resource.

7) PATCH -Applies partial modifications to a resource.

8) TRACE - Performs a message loop-back test along the path to the target resource.

9) CONNECT- Establishes a network connection to the target resource.

EXAMPLE:

```
  request->send(200, "text/html", html);
  });

  // Route to provide sensor values as JSON
  server.on("/sensorValues", HTTP_GET, [](AsyncWebServerRequest *request){
    String sensorValues = "{\"temperature\": " + String(dht.readTemperature(), 2) + ", \"moisture\": " + String(analogRead(A0)) + "}";
    request->send(200, "application/json", sensorValues);
  });

  // Route to provide flame status
  server.on("/status", HTTP_GET, [](AsyncWebServerRequest *request){
    String flameStatus = (digitalRead(34) == LOW) ? "Yes" : "No";
    request->send(200, "text/plain", flameStatus);
  });
```

4.7 SMTP PROTOCOL

SMTP, or Simple Mail Transfer Protocol, is a widely used network protocol for the transmission of electronic mail (email) between computers. It's the protocol responsible for sending emails from a client to a server or between servers. Here's a brief explanation of how SMTP works:

1. **Communication Flow:**

- **Client-Server Interaction:** SMTP operates on a client-server model. An email client (like Outlook or Thunderbird) communicates with an SMTP server to send emails.
- **Server-to-Server Communication:** When an email is sent from one domain to another, the SMTP server of the sender communicates with the SMTP server of the recipient.

2. **Port and Security:**
    - **Port Number:** SMTP typically uses port 25 for unencrypted communication and port 587 for encrypted communication (STARTTLS). Port 465 is also sometimes used for secure SMTP over SSL/TLS.
    - **Security:** In modern email systems, encryption (SSL/TLS) is often used to secure the communication between the email client and the server, preventing eavesdropping.
3. **Commands and Responses:**
    - **Commands:** The client issues commands to the server to send an email, specify recipients, etc. Common commands include EHLO (identify the client), MAIL FROM (specify the sender), RCPT TO (specify the recipient), DATA (start message transmission), etc.
    - **Responses:** The server responds to each command, indicating success or providing error information.
4. **Message Format:**

- **Header and Body:** The email message is divided into two parts: the header and the body. The header contains metadata like sender, recipient, subject, etc., while the body contains the actual message.
- **Termination:** The end of the message is indicated by a special sequence (CRLF.CRLF).

5. **Relaying and Forwarding:**
    - **Relaying:** SMTP servers can relay messages to other servers if the recipient is not on the same server. This facilitates email communication across different domains.
    - **Forwarding:** Servers may also forward emails to the next server in the destination domain until the email reaches its final destination.
6. **Authentication:**
    - **Authentication Methods:** To prevent unauthorized access, SMTP servers often require authentication. Common authentication methods include username/password, and more secure methods like OAuth.

CHAPTER-5

WORKING OF THE SYSTEM

5.1 Circuit Diagram

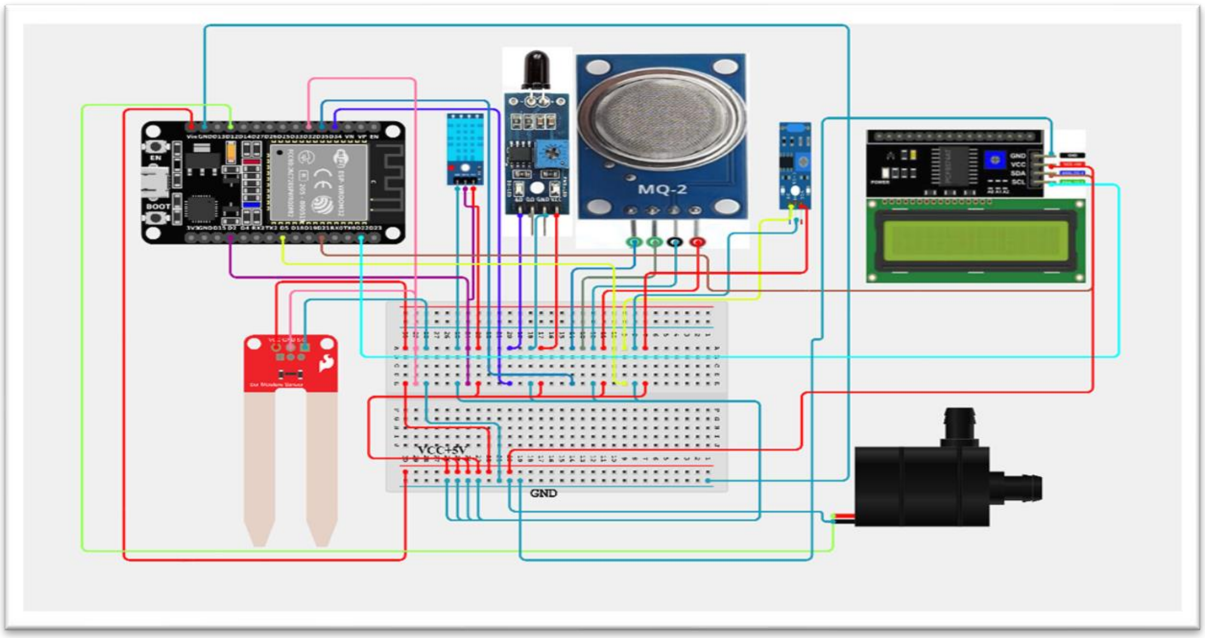


Fig 5.1 CIRCUIT DIAGRAM OF THE SYSTEM

The circuit diagram features a NodeMCU ESP32 microcontroller at its core, serving as the central processing unit. Connected to the NodeMCU are various sensors, including a DHT11 for temperature and humidity, a flame sensor for fire detection, an MQ2 gas sensor for detecting multiple gases, a SW-420 vibration sensor for motion detection, and a soil moisture sensor for measuring soil humidity. Additionally, a 5V water pump is integrated into the system. The output data from these sensors is processed by the NodeMCU and displayed on a 16x2 LCD using an I2C module, providing a user-friendly interface for monitoring environmental conditions. The circuit is designed to trigger the water pump based on sensor inputs, creating an automated system for watering plants or alerting users to potential hazards like fire or gas leaks. The integration of these components showcases a versatile and interconnected IoT (Internet of Things) system with applications in smart agriculture and environmental monitoring.

5.2 Working of the System

The home monitoring system project integrates an ESP32 microcontroller with an array of sensors including a gas sensor, flame sensor, earthquake sensor, soil moisture sensor, temperature, and humidity sensor, all aimed at ensuring comprehensive environmental monitoring within a household. The system is augmented with a 16x2 LCD display to provide offline data visualization. Additionally, it features a user-friendly web interface accessible wirelessly from anywhere via the internet, enabling users to monitor the real-time data remotely.

Upon power-up, the ESP32 initializes and establishes a connection to the internet using Wi-Fi connectivity. Once connected, the device starts reading data from the various sensors deployed across the household environment. The gas sensor and flame sensor continuously monitor for any signs of potential hazards such as gas leaks or fire outbreaks. Meanwhile, the earthquake sensor remains vigilant, detecting any seismic activity in the vicinity. The soil moisture sensor tracks soil hydration levels, crucial for maintaining healthy indoor plants. Concurrently, the temperature and humidity sensor provides insights into the indoor climate, ensuring comfort and well-being.

The collected sensor data is processed and displayed on the 16x2 LCD screen in real-time, allowing occupants to quickly assess the environmental conditions offline. Simultaneously, the device transmits this data to a dedicated web server where it is accessible through a user interface. Users can remotely monitor the environmental parameters and receive timely updates via email alerts, enhancing security and peace of mind. In the event of abnormal readings, such as high gas levels, fire, seismic activity, or unfavorable soil moisture levels, the system triggers automatic email alerts to notify users promptly. This proactive approach empowers homeowners to take necessary precautions or actions to mitigate risks and ensure the safety and well-being of their household.

In summary, the home monitoring system seamlessly integrates hardware components, sensor technologies, and internet connectivity to provide comprehensive environmental monitoring and remote accessibility. Through its intuitive user interface and alerting mechanisms, it offers users peace of mind by enabling them to stay informed and proactive in safeguarding their home environment.

CHAPTER – 6

INTERFACING AND TESTING

6.1 MICRO-CONTROLLER – DHT11 SENSOR INTERFACING

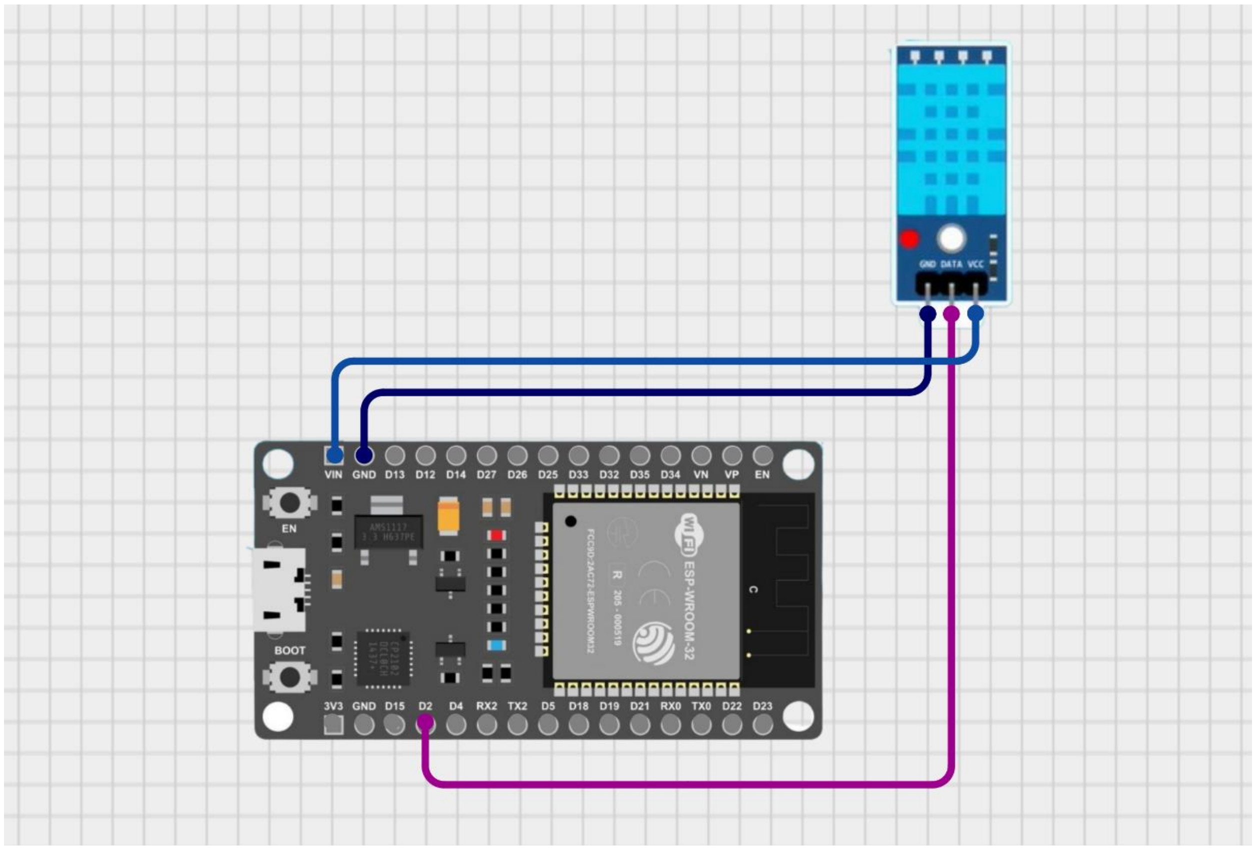

GND DATA VCC
VIN GND D13 D12 D14 D27 D26 D25 D33 D32 D35 D34 VN VP EN
3V3 GND D15 D2 D4 RX2 TX2 D5 D18 D19 D21 RX0 TX0 D22 D23
EN
BOOT
ESP-WROOM-32

Fig 6.1 : DHT-11 interfacing with ESP32

The ESP32 microcontroller seamlessly interfaces with the DHT11 sensor, with its data pin connected to D2. Leveraging software libraries such as Adafruit's, the ESP32 efficiently captures temperature and humidity data from the DHT11 sensor. The DHT11 sensor provides a temperature range of 0°C to 50°C with an accuracy of ±2°C and a humidity range of 20% to 90% RH with an accuracy of ±5%. This data, spanning a wide range of environmental conditions, can be transmitted to a Raspberry Pi for further analysis and action via protocols like MQTT or HTTP. This integration facilitates the creation of robust IoT applications, enabling real-time monitoring and automated responses tailored to diverse environmental parameters.

6.2 MICRO-CONTROLLER – MQ2 SENSOR INTERFACING

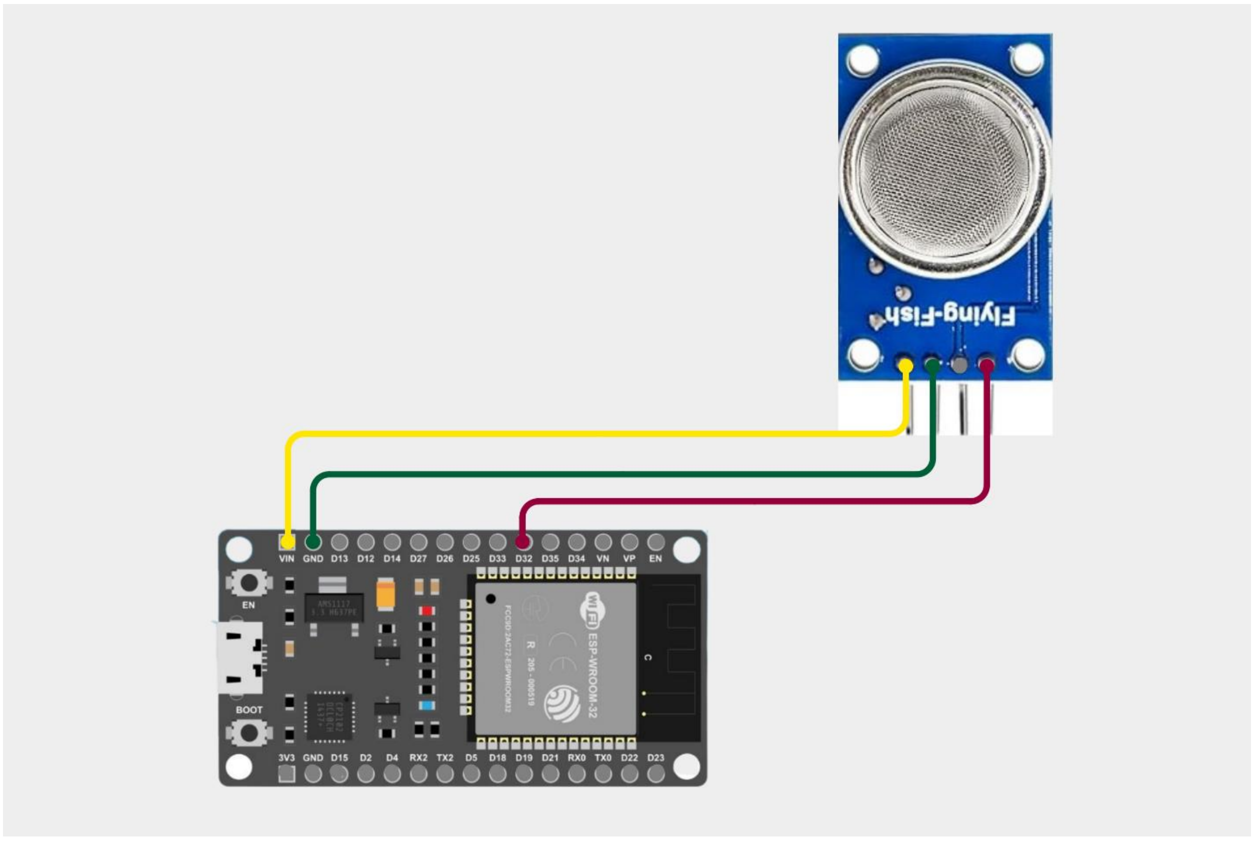


Fig 6.2: MQ2 sensor interfacing with ESP32

The ESP32 microcontroller seamlessly interfaces with the MQ2 gas sensor, utilizing GPIO pins for communication. The MQ2 sensor detects various gases such as LPG, propane, methane, alcohol, and smoke. Its detection range covers concentrations from 300 to 10,000 ppm for LPG, 200 to 10,000 ppm for propane, and 100 to 10,000 ppm for methane, with sensitivity adjustments possible through onboard potentiometers. Employing appropriate software libraries, like those available for Arduino, the ESP32 efficiently captures gas concentration data from the MQ2 sensor. This data, vital for ensuring indoor air quality and detecting potential hazards, can be transmitted to a Raspberry Pi for further processing using communication protocols such as MQTT or HTTP. This integration empowers IoT applications to monitor and respond to gas concentrations in real-time, enhancing safety and environmental awareness.

6.3 MICRO-CONTROLLER – SW-420 SENSOR INTERFACING

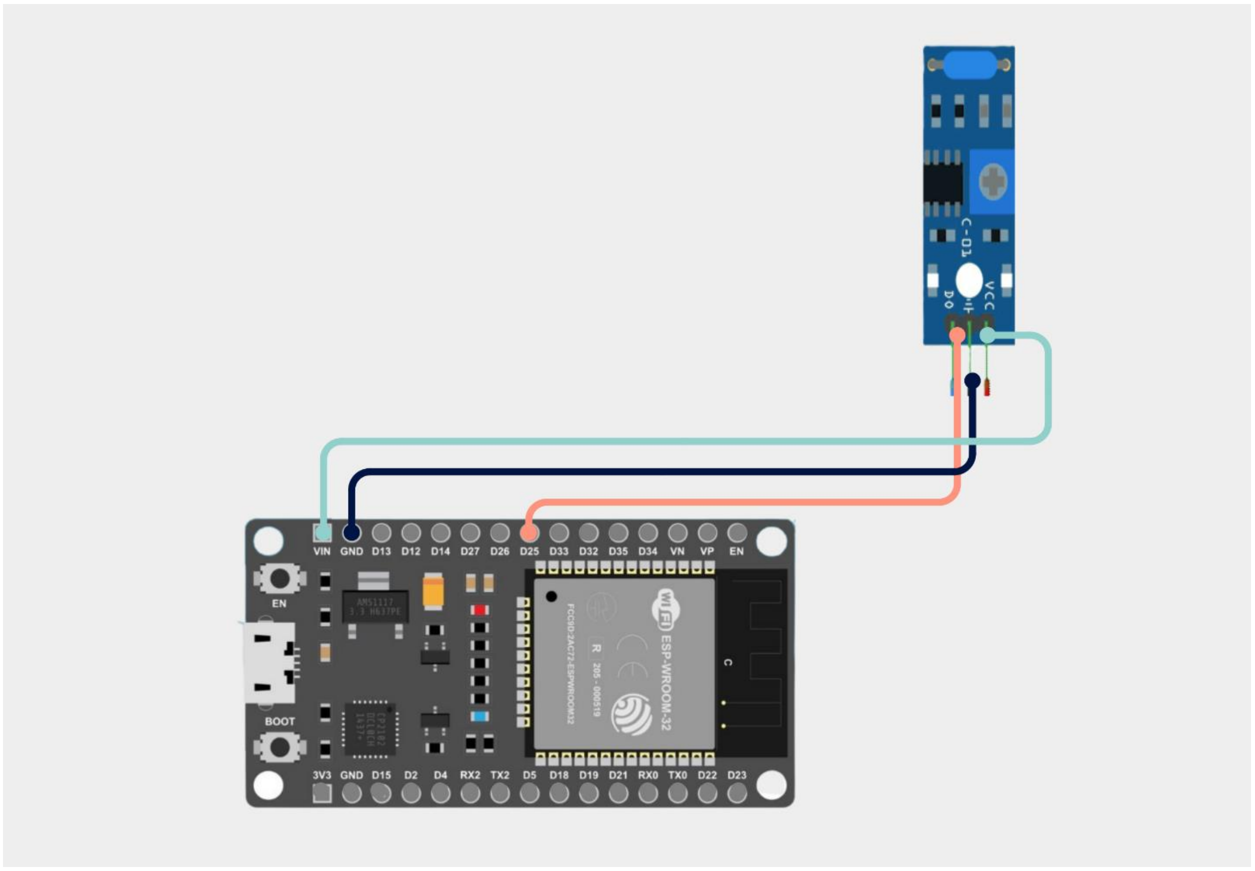


Fig 6.3 : sw-420 vibration sensor interfacing with ESP32

The ESP32 microcontroller seamlessly interfaces with the SW-420 vibration sensor, connecting the sensor's output to a GPIO pin for data acquisition. The SW-420 sensor detects vibrations and shocks, with a wide range of sensitivity adjustments. Its detection capabilities make it suitable for applications ranging from security systems to structural health monitoring. Leveraging software libraries compatible with the ESP32, such as those provided by Arduino, the microcontroller efficiently captures vibration data from the SW-420 sensor. This data, indicative of changes in the surrounding environment or equipment condition, can be transmitted to a Raspberry Pi for further analysis or action through communication protocols like MQTT or HTTP. By integrating the SW-420 vibration sensor with the ESP32, IoT applications gain the ability to monitor and respond to vibration events in real-time, enhancing safety and equipment maintenance practices.

6.4 MICRO-CONTROLLER – FLAME SENSOR INTERFACING

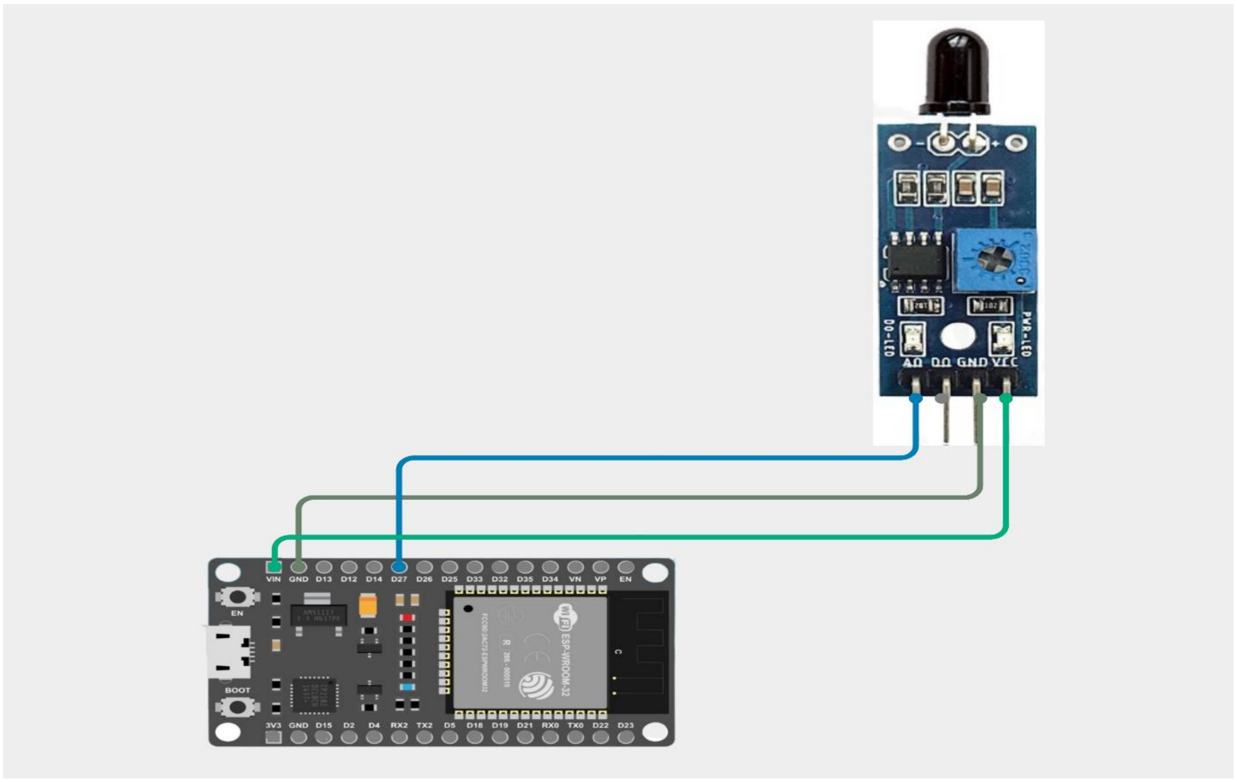
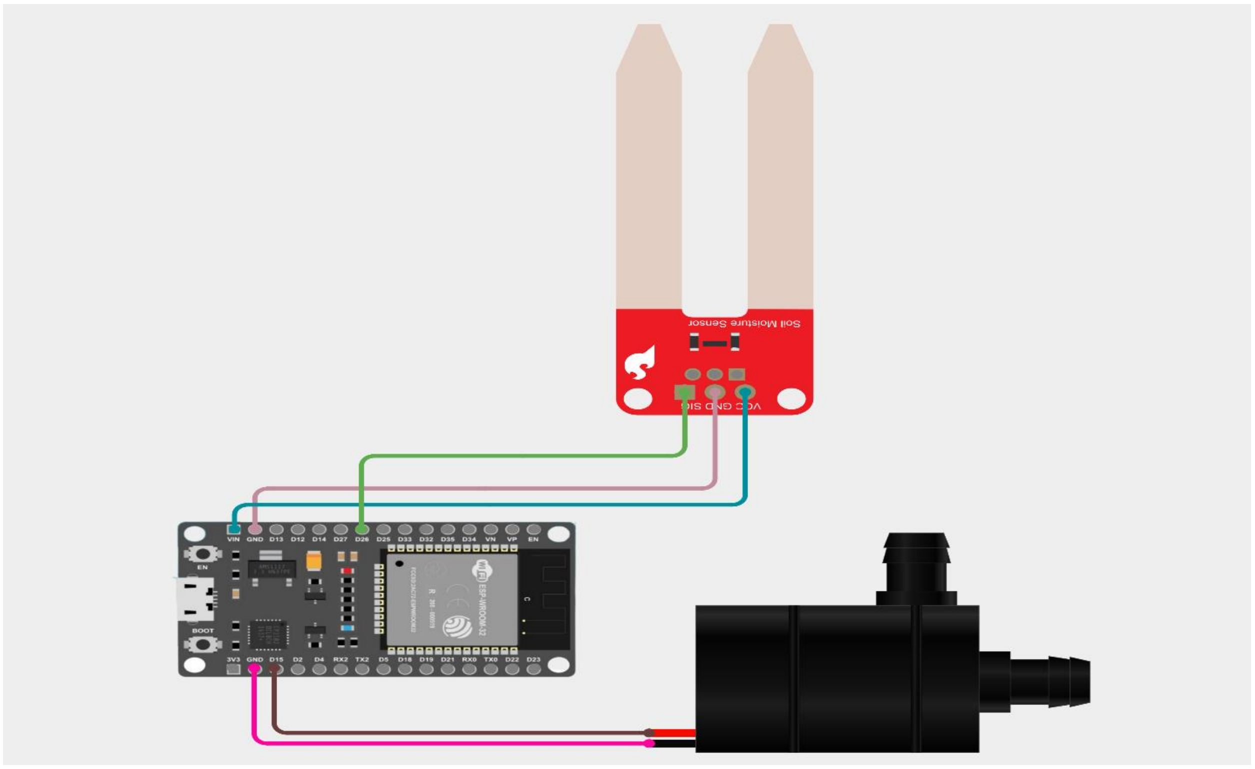


Fig 6.4 : flame sensor interfacing with ESP32

The ESP32 microcontroller seamlessly interfaces with the IR flame sensor, establishing communication through GPIO pins, typically utilizing analog or digital signals for data transmission. The IR flame sensor is designed to detect the presence of flames or fire by sensing infrared radiation emitted by flames. Its sensitivity can be adjusted to detect flames within a specific range, making it suitable for fire detection applications in various environments. By incorporating compatible software libraries, such as those tailored for Arduino, the ESP32 efficiently captures flame detection data from the IR sensor. This data, crucial for early fire detection and prevention, can be transmitted to a Raspberry Pi for further processing and action via communication protocols like MQTT or HTTP. Integrating the IR flame sensor with the ESP32 empowers IoT applications to monitor and respond to potential fire hazards in real-time, enhancing safety and minimizing risks of property damage.

6.5 MICRO-CONTROLLER – SOIL MOISTURE SENSOR INTERFACING



Soil Moisture Sensor
VCC GND SIG
VIN GND D13 D12 D14 D27 D26 D25 D33 D32 D35 D34 VN VP EN
3V3 GND D15 D2 D4 RX2 TX2 D5 D18 D19 D21 RX0 TX0 D22 D23
EN
BOOT
ESP-WROOM-32

Fig 6.5 : soil moisture sensor interfacing with ESP32

The ESP32 microcontroller seamlessly interfaces with the soil moisture sensor and water pump, utilizing GPIO pins to control the water pump's activation based on soil moisture readings. The soil moisture sensor measures the moisture content in the soil, providing data crucial for efficient plant watering. Once the soil moisture level falls below a predefined threshold, signaling dryness, the ESP32 triggers the 5V water pump to initiate watering. This integration ensures timely and automated watering of plants, optimizing growth conditions. By incorporating compatible software libraries, such as those supported by Arduino, the ESP32 efficiently reads soil moisture data from the sensor and controls the water pump accordingly. Additionally, this data can be transmitted to a Raspberry Pi for further analysis or monitoring, facilitating intelligent irrigation systems. Integrating the soil moisture sensor with the ESP32 and water pump enables IoT applications to maintain optimal soil moisture levels, promoting healthy plant growth while conserving water resources effectively.

CHAPTER-7

HIGHLIGHTS

7.1 ADVANTAGES

1. Increased Security
2. Real-Time Data Accessibility
3. Safety Features
4. User-Friendly Interface
5. Wireless Connectivity
6. Optimal Plant Care

7.2 APPLICATIONS

1. Home Security
2. Environmental Monitoring
3. Smart Agriculture
4. Building Management Systems
5. Disaster Management

RESULT

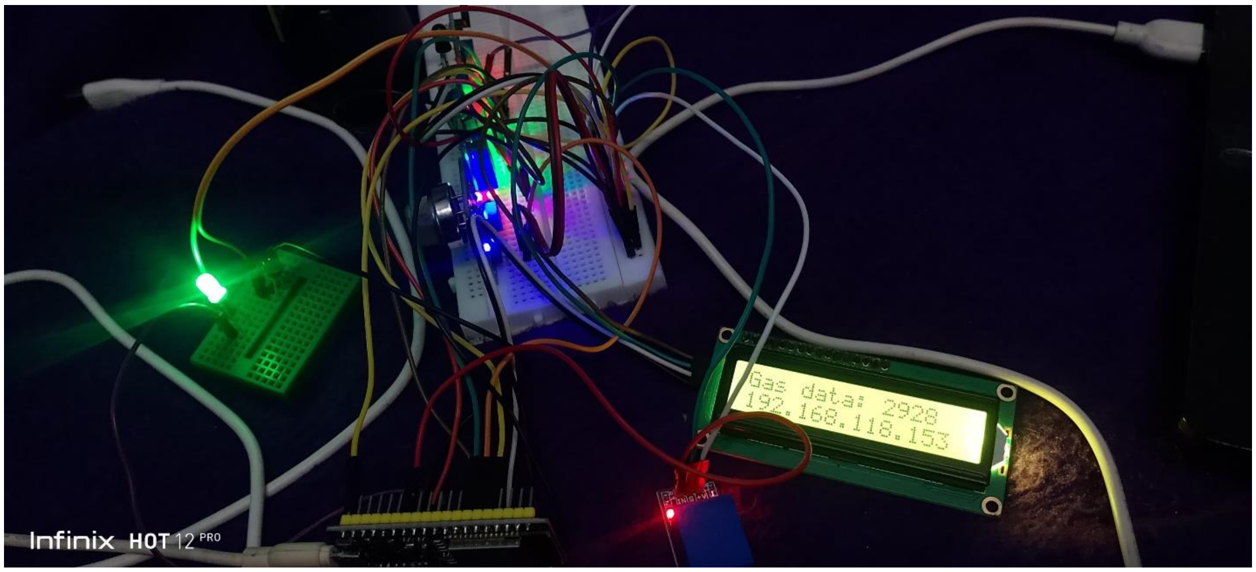


Fig (a): Initialization and Working of Module

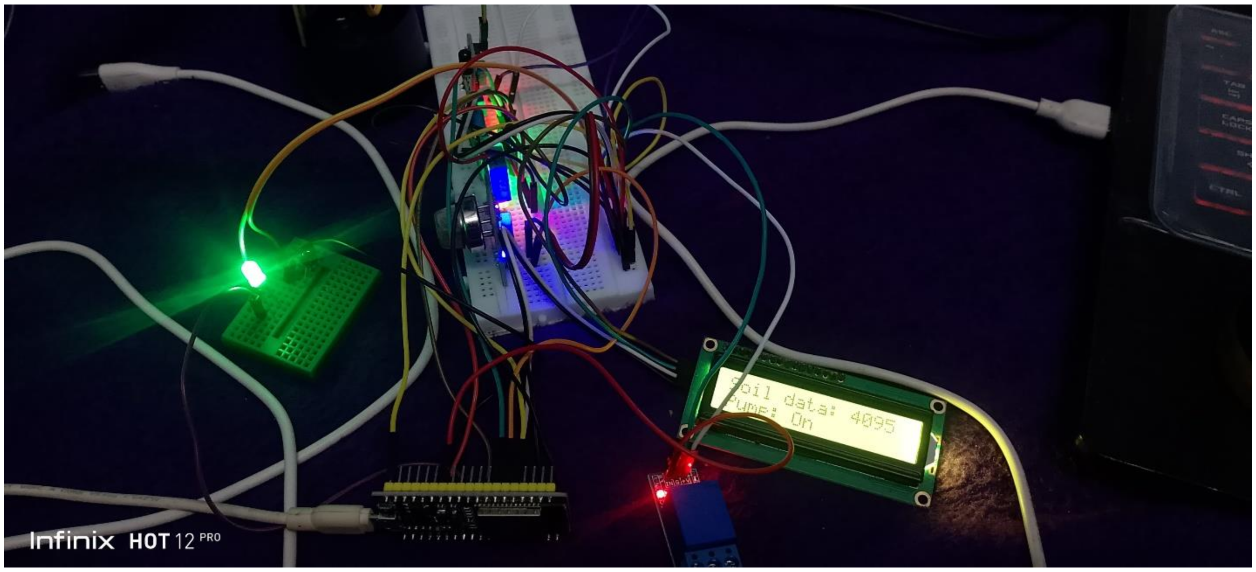


Fig (b): Initialization and Working of Module

CONCLUSION

In addition to its core functionalities, this home monitoring system offers a multifaceted approach to enhancing the safety and well-being of your household. By leveraging the capabilities of the ESP32 module and a diverse array of sensors, it provides a comprehensive insight into various aspects of your home environment.

Beyond simply monitoring temperature, humidity, and soil moisture, this system delves deeper, offering real-time detection of potential hazards such as earthquakes, gas leaks, and fire outbreaks. This proactive approach to safety not only safeguards your property but also ensures the protection of your loved ones. The inclusion of motion sensors adds another layer of security, enabling you to monitor activity within your home even when you're away.

Moreover, the system's ability to display this information in real-time, both on a dedicated website and a local screen, empowers users with actionable insights. Whether you're at home or halfway around the world, you can stay informed about your home's conditions and make informed decisions accordingly. This level of accessibility and convenience epitomizes the essence of a truly smart home solution.

Furthermore, the system's versatility extends beyond mere monitoring. With the capability to send alerts via email, it provides timely notifications in the event of any anomalies, allowing for swift intervention and mitigation. This proactive approach not only enhances security but also promotes peace of mind, knowing that your home is being actively monitored and protected.

In essence, this home monitoring system represents more than just a collection of sensors and devices—it embodies the convergence of technology and safety, offering a holistic solution for modern homeowners. Its ease of use, coupled with its robust feature set, makes it a cornerstone of any smart home ecosystem, ensuring that your home remains safe, secure, and conducive to healthy living.

FUTURE SCOPE

Expanding the future scope of HOME MONITERING SYSTEM holds immense potential for advancing its capabilities and enhancing its utility. By integrating an MQTT server into the architecture, you'll establish a robust foundation for centralized data storage and analysis. This server will serve as a repository for the sensor data collected by the ESP32, facilitating organized data management and enabling seamless integration with other systems or applications. Furthermore, implementing data analysis algorithms will unlock valuable insights from the collected data, empowering informed decision-making and facilitating predictive analytics. Security enhancements, including encryption, authentication protocols, and role-based access control, will ensure that sensitive data remains secure and accessible only to authorized users. With a user authentication system in place, users will be able to securely access the data through a personalized web interface or mobile application, tailored to their specific roles and permissions. Real-time alerts and notifications will provide users with proactive insights, enabling timely responses to critical events or anomalies detected by the sensors. By embracing these future enhancements, HOME MONITERING SYSTEM will evolve into a comprehensive IoT solution, empowering users with actionable insights and optimizing operational efficiency.

APPENDIX

```
#include <WiFi.h>

#include <ESPAsyncWebServer.h>

#include <ESP32_MailClient.h>

#include <DHT.h>

#include <Wire.h>

#include <LiquidCrystal_I2C.h>


const char* ssid = "Pavan";

const char* password = "Pavan123";


#define DHTPIN 2          // Pin where the DHT11 is connected

#define DHTTYPE DHT11     // DHT11 sensor type

DHT dht(DHTPIN, DHTTYPE);

LiquidCrystal_I2C lcd(0x27, 16, 2);


void sendEmailNotification(bool flameDetected);

void sendEmailNotification(float temperature);

void sendEmailNotificationVibration(bool vibrationDetected);// Function
declaration for sending vibration notification

void sendEmailNotificationGas(int gasdetected);

const char* emailSubjectGas = "Gas Detected!";

const char* emailBodyGas = "Gas Detected! Dangerous gas levels detected! Take
necessary precautions.";


const char* emailSenderAccount = "homemoniteringsystem@gmail.com";
const char* emailSenderPassword = "dsivznmdtvpcazjp";
const char* smtpServer = "smtp.gmail.com";
const int smtpServerPort = 465;
const char* emailSubject = "ALERT! High Temperature";
const int VIBRATION_PIN = 5;  // Pin connected to the vibration sensor
const int gasSensorPin = 35; // Analog pin for MQ2 gas sensor
const int threshold = 2500; // Adjust the threshold as needed
const int greenLedPin = 4; // Pin for green LED
const int redLedPin = 15;   // Pin for red LED

String pumpStatus; // Initialize pump status
String vibrationStatus = "LOW"; // Initialize vibration status
int moistureLevel = 0; // Initialize moisture level
const int flameDigitalPin = 34;  // Replace with the actual digital pin connected to
the flame sensor
bool flameDetected = false;
float thresholdValue = 35.0;
const int moisturePin = 32;   // Replace with the actual analog pin connected to the
moisture sensor
const int pumpPin = 12;       // Replace with the actual digital pin connected to the
water pump
int moistureThreshold = 3000;  // Adjust this threshold based on your moisture
sensor
string gasStatus;

AsyncWebServer server(80);
```

```
String html = R"(
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Home Monitoring System</title>

    <style>
        body {
            margin: 0;
            font-family: 'Arial', sans-serif;
            background: linear-gradient(to right, #ff8c00, #ff007f); /* Gradient
background */
            color: #ffffff; /* Text color */
        }

        .container {
            margin: 20px;
            text-align: center;
        }

        /* Navbar styles */
        .navbar {
            display: flex;
            justify-content: space-between;
            background-color: rgba(0, 0, 0, 0.7); /* Navbar background color with
transparency */
            height: 70px;
            border: 1px solid white;
        }

        .left_bar, .right_bar {
            display: flex;
            align-items: center;
        }

        .a-element {
            padding: 10px;
            text-decoration: none;
            margin: 2.5px;
            color: #ffffff;
            font-size: 14px;
            font-weight: bold;
            border-radius: 5px;
            transition: background-color 0.3s ease-in-out; /* Smooth color transition */
        }

        .a-element:hover {
            background-color: rgba(255, 255, 255, 0.2); /* Semi-transparent white on
hover */
        }
```

```
/* Info section styles */
.info-section {
    margin-bottom: 30px;
}

/* Sensor data styles */
.sensor-data {
    font-size: 35px;
    margin-top: 20px;
}

/* Responsive styles */
@media screen and (max-width: 600px) {
    .navbar {
        flex-direction: column;
        align-items: center;
        height: auto;
    }

    .right_bar {
        margin-top: 10px;
    }
}
```

```
        /* Keyframe animation for temperature */
        @keyframes alertAnimation {
            0%, 100% {
                background-color: transparent;
            }
            50% {
                background-color: #ff0000; /* Red */
            }
        }

        /* Apply animation to temperature */
        #temperature {
            animation: alertAnimation 2s infinite;
            border-radius: 5px;
        }

        .guide-name {
            color: #0af7ff; /* OrangeRed */
            font-size: 24px;
            font-weight: bold;
        }
    </style>
</head>
<body>
    <div class="navbar">
```

```
        <div class="left_bar">
            <h3 class="title">Home Monitoring System</h3>
        </div>
        <div class="right_bar">
            <a href="https://bit.ly/homemoniteringsystem" class="a-element"
target="_blank">Home</a>
            <a href="https://pavan-1522.github.io/home-
monitering_system/about.html" class="a-element" target="_blank">About</a>
            <a href="https://pavan-1522.github.io/home-
monitering_system/service.html" class="a-element" target="_blank">Our
Project</a>
            <a href="https://bit.ly/esp32home" class="a-element"
target="_blank">Panel</a>
            <a href="https://pavan-1522.github.io/home-monitering_system/why.html"
class="a-element" target="_blank">Why This</a>
            <a href="https://pavan-1522.github.io/home-monitering_system/team.html"
class="a-element" target="_blank">Our Team</a>
        </div>
    </div>


    <div class="container">
        <h1>Comprehensive Home Monitoring System with Environmental Sensing
using ESP32</h1>
        <div class="info-section">
            <h2 id="title">Chaitanya Engineering College</h2>
            <img
src="https://res.cloudinary.com/dudz8iroq/image/upload/v1709373870/CECLogo_
p0uu7w.jpg" alt="no image">
```

```
            <h3 id="depart">Department of Electronics and Communication
Engineering</h3>
            <h4 id="guide">Under The Guidance Of: <span class="guide-name">Ms.
B. Radha Devi, M.Tech</span></h4>
        </div>
        <hr>
        <!-- Your container HTML -->
        <div class="sensor-data">
            <p>Temperature: <span id="temperature"> %TEMPERATURE% </span>
&deg;C</p>
            <p>Humidity: <span id="humidity">%humidity%</span> %</p>
            <p>Flame Detected: <span id="flameStatus"> %FLAME% </span></p>
            <p>Moisture Level: <span
id="moistureStatus">%MOISTURE%</span></p>
            <p>Water Pump: <span id="pumpStatus">%PUMP%</span></p>
            <p>Earthquake Status: <span
id="earthquakeStatus">%VIBRATION%</span></p>
            <p>Gas Sensor Status: <span id="gasSensorStatus">%GAS%</span></p>
        </div>
    </div>
    <script>
      function refreshPage() {
          setTimeout(() => {
              location.reload();
          }, 1000); // Refresh every 1000 milliseconds (1 second)
        }
        // Call the function initially
        refreshPage();
    </script>
</body>
</html>
)";

String processor(const String& var) {
    if (var == "TEMPERATURE") {
        // Read the temperature from the DHT sensor
        float temperature = dht.readTemperature();
        // Convert temperature to string with two decimal places
        return String(temperature, 2);
    }
    // Add more variables as needed
    return String();
}

void setup() {
    Serial.begin(115200);

    // Connect to Wi-Fi
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
         delay(500);
         digitalWrite(greenLedPin, LOW); // Ensure green LED is off while
connecting
         digitalWrite(redLedPin, HIGH);  // Turn on red LED to indicate no
connection
         delay(500);
         digitalWrite(redLedPin, LOW);   // Turn off red LED
         Serial.println("Connecting to WiFi...");
        }
    Serial.println("Connected to WiFi");
    digitalWrite(greenLedPin, HIGH);
    Serial.println(WiFi.localIP());

    dht.begin();
    pinMode(flameDigitalPin, INPUT);
    pinMode(moisturePin, INPUT);
    pinMode(pumpPin, OUTPUT);
    digitalWrite(pumpPin, LOW);
    pinMode(gasSensorPin, INPUT);
    pinMode(VIBRATION_PIN, INPUT); // Set the vibration sensor pin as input
    pinMode(greenLedPin, OUTPUT);
    pinMode(redLedPin, OUTPUT);

   server.on("/", HTTP_GET, [](AsyncWebServerRequest *request) {
    String dynamicHtml = html;
    dynamicHtml.replace("%TEMPERATURE%", String(dht.readTemperature(),
2).c_str());
```

```
    dynamicHtml.replace("%humidity%", String(dht.readHumidity()).c_str()); //
Update humidity value
    dynamicHtml.replace("%MOISTURE%", String(moistureLevel).c_str());
    dynamicHtml.replace("%GAS%", String(gasStatus.c_str()));
    dynamicHtml.replace("%FLAME%", flameDetected ? "Yes" : "No");
    dynamicHtml.replace("%VIBRATION%", vibrationStatus.c_str());
    dynamicHtml.replace("%PUMP%", pumpStatus.c_str());
    request->send(200, "text/html", dynamicHtml);
});

    server.on("/", HTTP_GET, [](AsyncWebServerRequest *request) {
        // Serve HTML page
        request->send(200, "text/html", html);
    });

    server.begin();
  lcd.begin();
  lcd.backlight();

  lcd.print("ESP32 Monitoring");
  lcd.setCursor(0, 1);
  lcd.print("System");
  delay(2000);
  lcd.clear();
}
```

```
void loop() {
   // checking wifi status
  if (WiFi.status() != WL_CONNECTED) {
    // Wi-Fi disconnected, turn on red LED
    digitalWrite(greenLedPin, LOW);
    digitalWrite(redLedPin, HIGH);
  } else {
    // Wi-Fi connected, turn off red LED
    digitalWrite(redLedPin, LOW);
  }
    // Reading sensor data
    float temperature = dht.readTemperature();
    moistureLevel = analogRead(moisturePin);
    int gasValue = analogRead(gasSensorPin);

    // Check if temperature exceeds the threshold value
    if (temperature > thresholdValue) {
        // Send email notification
        sendEmailNotification(temperature);
    }

    delay(1000); // Wait for 1 second before reading sensor again

    flameDetected = digitalRead(flameDigitalPin) == LOW;
```

```
delay(500);  // delay half second to read sensor data
if (flameDetected) {
   sendEmailNotification(flameDetected);
}
else {
   Serial.println("No flame detected");
}

// Moisture sensor starts
moistureLevel = analogRead(moisturePin);
if (moistureLevel > moistureThreshold) {
   // Turn on the water pump
   digitalWrite(pumpPin, HIGH);
   pumpStatus = "On";
}
else {
   // Turn off the water pump
   digitalWrite(pumpPin, LOW);
   pumpStatus = "Off";
}

delay(1000);  // Adjust the delay as needed

// Vibration sensor starts
bool vibrationDetected = digitalRead(VIBRATION_PIN) == HIGH;
```

```
  if (vibrationDetected) {
    // Send email notification for vibration detected
    sendEmailNotificationVibration(vibrationDetected);
    vibrationStatus = "HIGH";
  }
  else {
    vibrationStatus = "LOW";
  }

  // MQ2 sensor starts
  Serial.println("Gas sensor value: " + String(gasValue));
  int gasDetected = gasValue > threshold;
  if (gasValue > threshold) {
    gasStatus="detected";
    sendEmailNotificationGas(gasDetected);
    Serial.println("Email sent!");
    delay(5000); // Wait for a minute to avoid sending multiple emails in a short
time
  }else{
   gasStatus="not deteceetd";
  }

   // Display flame status
 lcd.setCursor(0, 0);
 lcd.print("Temp: ");
 lcd.print(temperature);
 lcd.print("c");

 // Display temperature
 lcd.setCursor(0, 1);
 lcd.print("Hum: ");
 lcd.print(dht.readHumidity());
 lcd.print(" C");

 delay(4000);
 lcd.clear();

 // Display humidity
 lcd.setCursor(0, 0);
 lcd.print("Flame: ");
 lcd.print(flameDetected ? "Yes" : "No");

 // Display earthquake status
 lcd.setCursor(0, 1);
 lcd.print("Earthquake: ");
 lcd.print(vibrationStatus);

 delay(4000);
 lcd.clear();

 // Display soil moisture
```

```
  lcd.setCursor(0, 0);
  lcd.print("Soil data: ");
  lcd.print(moistureLevel);

  // Display pump status
  lcd.setCursor(0, 1);
  lcd.print("Pump: ");
  lcd.print(pumpStatus);

  delay(4000);
  lcd.clear();

    // Display MQ2 value
  lcd.setCursor(0, 0);
  lcd.print("Gas data: ");
  lcd.print(gasValue);

  // Display pump status
  lcd.setCursor(0, 1);
  // lcd.print("ip: ");
  lcd.print(WiFi.localIP());
  delay(4000);
  lcd.clear();
}
```

```
void sendEmailNotification(bool flameDetected) {
   SMTPData smtpData;
   smtpData.setLogin(smtpServer, smtpServerPort, emailSenderAccount,
emailSenderPassword);
   smtpData.setSender("ESP32_Fire_Alert", emailSenderAccount);
   smtpData.setPriority("High");
   smtpData.setSubject(emailSubject);
   String emailMessage = "Flame detected, hurry up DIAL: 101 for help";
   smtpData.setMessage(emailMessage, true);
   smtpData.addRecipient("pavankumarmadeti143@gmail.com");
   if (!MailClient.sendMail(smtpData)) {
      Serial.println("Error sending Email, " + MailClient.smtpErrorReason());
   }
   else {
      Serial.println("Email sent successfully for fire");
   }
   smtpData.empty();
}

void sendEmailNotification(float temperature) {
   SMTPData smtpData;
   smtpData.setLogin(smtpServer, smtpServerPort, emailSenderAccount,
emailSenderPassword);
   smtpData.setSender("ESP32_Temperature_Alert", emailSenderAccount);
   smtpData.setPriority("High");
   smtpData.setSubject(emailSubject);
```

```
  String emailMessage = "High temperature alert! Current temperature is " +
String(temperature) + " °C.";
  smtpData.setMessage(emailMessage, true);
  smtpData.addRecipient("pavankumarmadeti143@gmail.com");
  if (!MailClient.sendMail(smtpData)) {
    Serial.println("Error sending Email, " + MailClient.smtpErrorReason());
  }
  else {
    Serial.println("Email sent successfully for temperature");
  }
  smtpData.empty();
}

void sendEmailNotificationVibration(bool vibrationDetected) {
  SMTPData smtpData;
  smtpData.setLogin(smtpServer, smtpServerPort, emailSenderAccount,
emailSenderPassword);
  smtpData.setSender("ESP32_Vibration_Alert", emailSenderAccount);
  smtpData.setPriority("High");
  smtpData.setSubject("ALERT! Vibration Detected");
  String emailMessage = "Vibration detected! Check the home monitoring system
for details.";
  smtpData.setMessage(emailMessage, true);
  smtpData.addRecipient("pavankumarmadeti143@gmail.com");
  if (!MailClient.sendMail(smtpData)) {
    Serial.println("Error sending Email, " + MailClient.smtpErrorReason());
  }
  else {
    Serial.println("Email sent successfully for vibration");
  }
  smtpData.empty();
}

void sendEmailNotificationGas(int gasDetected) {
  SMTPData smtpData;
  smtpData.setLogin(smtpServer, smtpServerPort, emailSenderAccount,
emailSenderPassword);
  smtpData.setSender("ESP32_GAS_Alert", emailSenderAccount);
  smtpData.setPriority("High");
  smtpData.setSubject("ALERT! GAS Detected");
  String emailMessage = "GAS detected! Check the home monitoring system for
details.";
  smtpData.setMessage(emailMessage, true);
  smtpData.addRecipient("pavankumarmadeti143@gmail.com");
  if (!MailClient.sendMail(smtpData)) {
    Serial.println("Error sending Email, " + MailClient.smtpErrorReason());
  }
  else {
    Serial.println("Email sent successfully for GAS");
  }
  smtpData.empty();
}
```