Smart Water Heater

1st Kunal Pravin Patil
(2322223) dept. of Instrumentation Government of college of Engineering Jalgaon
2322223@gcoej.ac.in

2nd Tanuja Anant Patil
(2322225) Dept of Instrumentation Government of college of Engineering Jalgaon
2322225@gcoej.ac.in

Guided By
Dr. Prashant J. Gaidhane
Dept. of Instrumentation Government college of Engineering, Jalgaon

***ABSTRACT*—This project presents the design and implementation of Smart Water Heater. Water quality and environmental factors have a significant impact on the efficiency and longevity of water heating systems. Traditional heating appliances, such as electric kettles, lack the ability to monitor or adapt to real-time conditions such as ambient temperature, humidity, and pressure. This can result in energy inefficiency, scaling, and potential wear over time.This project introduces a Smart Water Heater, implemented using an electric kettle integrated with intelligent sensing and automation. The system is built around an ESP32 microcontroller and utilizes a DS18B20 temperature sensor to monitor water temperature, a BMP sensor for capturing environmental parameters (humidity, air pressure, and outdoor temperature), a touch sensor for user interaction, an OLED display for real-time data visualization, and a relay to control the kettle's power. The smart control logic ensures optimized heating by considering environmental data, user preferences, and safety thresholds. This approach enhances energy efficiency, reduces scaling risks, and provides a user-friendly interface for modern household or lab applications.**

I. Introduction

Water heating is an essential utility in both domestic and industrial environments. Electric kettles are widely used for their convenience and speed, but traditional models operate in a binary on/off mode without regard for environmental conditions or water quality. This leads to inefficient power usage, possible overheating, and long-term wear such as limescale buildup and corrosion, especially in regions with hard water or varying ambient conditions.

To address these limitations, this project proposes the development of a Smart Water Heater using an electric kettle. The system is designed to optimize heating operations through intelligent automation and environmental awareness. At the heart of the system is the ESP32, as shown in Fig.1, a powerful microcontroller with built-in Wi-Fi and Bluetooth capabilities, allowing for potential future expansions such as remote monitoring or mobile app integration.

Sensors like the DS18B20 allow precise monitoring of the water temperature, while the BMP sensor collects ambient data, enabling adaptive heating based on current weather or room conditions. A touch sensor enables manual user input, and an OLED display presents real-time system status and readings. A relay module is used to switch the kettle on or off based on logic processed by the microcontroller.

This project not only aims to make water heating more efficient and eco-friendlier but also showcases the practical application of IoT (Internet of Things) principles in everyday appliances.

II. Component Used

1 ESP 32 Microcontroller: The **ESP32 (ESP-WROOM-32)** is a powerful and versatile microcontroller module developed by Espressif Systems. It is widely used in IoT (Internet of Things) applications due to its integrated Wi-Fi, Bluetooth (Classic and BLE), and extensive peripheral support. In this project, the ESP32 serves as the central processing unit that reads data from various sensors, processes it in real-time, controls the heating element via a relay, and updates the user interface on the OLED display. A low-power, Wi-Fi- and Bluetooth-enabled microcontroller that handles communication with the Blynk platform and control logic for the relay and sensor inputs.

Specifications:	
Operating Voltage	2.2 ~ 3.6V
Frequency range	2.4 ~ 2.5 GHz
Model Name	ESP- WROOM32
RAM Memory Installed Size	520 KB
CPU Speed	2.4 GHz
Connectivity via	Bluetooth, Wi-Fi

Fig. 1 ESP32

Use:

In the Smart Water Heater system, the ESP32 is responsible for:

- Reading temperature data from the DS18B20 (refer fig 2) sensor.
- Gathering environmental data (humidity, pressure, temperature) from the BMP sensor.
- Detecting user input via the touch sensor.
- Displaying real-time readings on the OLED screen.
- Controlling the relay to turn the kettle on or off based on programmed logic.

2.DS18B20 temperature sensor: The DS18B20 is a digital temperature sensor widely used in embedded systems for accurate and reliable temperature monitoring. It operates over a temperature range of -55°C to +125°C with an accuracy of ±0.5°C in the range of -10°C to +85°C. One of its key advantages is the use of the 1-Wire communication protocol, which allows data transmission over a single data line, minimizing wiring complexity and conserving GPIO pins on the microcontroller. The sensor supports programmable resolution from 9 to 12 bits, enabling users to balance between response time and precision as needed.

In the Smart Water Kettle project, the DS18B20 sensor plays a vital role in measuring the real-time temperature of the water inside the kettle. It is connected to the ESP32, which reads the sensor data and compares it to a predefined temperature threshold. Based on this comparison, the ESP32 controls a relay module that turns the electric kettle ON or OFF, ensuring that the water is heated to the desired level without overheating.

The live temperature readings are also displayed on the OLED screen, providing real-time feedback to the user. The DS18B20's digital output, high accuracy, and waterproof casing make it ideal for this application, where consistent and safe heating is essential.

Fig.2 DS18B20 Sensor

Specifications:	
Operating Voltage	5.5V
Accuracy	±0.5°C
Temperature Sensor	DS18B20
Cable Length	Varies

3. BMP280: The BMP280 as shown in fig 3 is a highly precise, low-power barometric pressure sensor developed by Bosch, commonly used in weather forecasting and environment-sensitive applications. It is capable of measuring atmospheric pressure and temperature, with high accuracy and fast response time. The sensor supports both I2C and SPI communication protocols, making it easy to interface with microcontrollers like the ESP32. In your Smart Water Kettle project, the BMP280 is used to monitor ambient temperature and air pressure, which can help in understanding the external environment that may influence the water heating process. For instance, changes in room temperature or altitude-related pressure can be taken into account to adjust heating logic for efficiency. The small size, low power consumption, and high resolution of the BMP280 make it a perfect fit for IoT-based smart appliances.

Specifications:	
Operating Voltage	3.3V
Communication Interface	I2C and SPI
Temperature Accuracy	±1.0°C

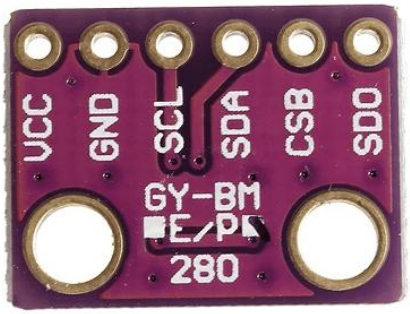


Fig.3 BMP280

4.Touch Sensor: A breadboard is a tool for constructing electronic In this project, two capacitive touch sensors are used to provide a user-friendly and intuitive control interface for the Smart Water Kettle. These sensors act as digital inputs to the ESP32 and are used to trigger different actions. The positive touch sensors shown in fig 4 is used to start or increase the target temperature, while the negative touch sensor is used to stop or decrease the temperature setting. This setup replaces traditional push-buttons with capacitive input, offering a more modern, sleek, and durable interface with no mechanical wear and tear.

When a user touches the positive sensor, the ESP32 detects a HIGH signal on the corresponding GPIO pin and begins or increases the heating process. Similarly, a touch on the negative sensor sends a HIGH signal to another GPIO pin, prompting the system to stop heating or lower the target threshold. This simple, dual-sensor design allows users to easily interact with the system without any complex interface or external app. It also complements the overall smart and efficient design of the water kettle system.

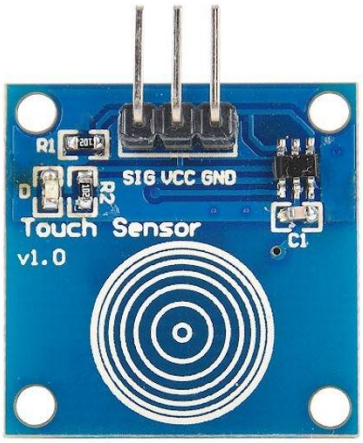


Fig.4 Touch Sensor

Specifications:	
Operating Voltage	2.0V to 5.5V
Interface ESP32	Digital GPIO pins on

5. OLED: The OLED as shown in fig 5 (Organic Light Emitting Diode) display is used in this project to provide real-time visual feedback to the user. Typically, a 0.96-inch 128×64-pixel I2C OLED display based on the SSD1306 driver is used due to its compact size, low power consumption, and excellent contrast. The display shows important information such as current water temperature (from the DS18B20 sensor), ambient temperature and pressure (from the BMP sensor), and system status messages like "Heating ON", "Target Reached", or "Standby".

The display enhances the user experience by making the system more interactive and informative without the need for additional external devices like smartphones or computers. The OLED module communicates with the ESP32 using the I2C protocol, requiring only two pins (SCL and SDA), which makes it simple to wire and ideal for embedded systems with limited GPIOs. Its ability to display crisp text and basic graphics also opens the door for future improvements, such as icons, bar graphs, or custom menus. The inclusion of the OLED screen makes the Smart Water Kettle system not only smarter but also more user-friendly and visually accessible.

Fig.5 OLED Display

Specifications:	
Display Type	OLED
Forward Voltage	3.3V

6. Relay: The relay as shown in fig 6 module is a critical component in the Smart Water Kettle project, enabling the ESP32 microcontroller to control high-voltage devices such as the heating element (in this case, the electric kettle). Since the ESP32 operates at low voltage (3.3V logic), it cannot directly switch on/off a 230V AC appliance. The relay acts as an electromechanical switch, allowing the ESP32 to control the kettle safely and efficiently through a digital output pin.

When the temperature reading from the DS18B20 sensor reaches the desired threshold, the ESP32 sends a signal to the relay module, which then turns off the power supply to the kettle, preventing overheating and saving energy. Likewise, it switches the kettle ON when heating is needed.
This automation improves both safety and energy efficiency, and forms the core of the heating control logic.

Fig.6 Relay Module

7.Kettle: The electric kettle used in this project is a common household device used to boil water quickly and easily. It is made from strong materials like stainless steel or safe plastic, which makes it durable and safe to use. Inside the kettle, there is a heating coil at the bottom that gets hot when electricity flows through it, which helps in boiling the water. The kettle used in this project has a capacity of 1.5 liters, which is suitable for regular use. It usually works on power between 600 to 2000 watts, so it boils water fast. The kettle also has a lid and a handle for safe and easy pouring. It includes a safety feature called thermal cut-off, which automatically turns off the heating if the water is already boiled or if there is no water inside.

Fig.7 Kettle

III.Working

The Smart Water Kettle is an IoT-based system designed for intelligent and efficient water heating. The ESP32 microcontroller acts as the brain of the system, collecting data from multiple sensors and controlling the kettle via a relay.

The DS18B20 sensor continuously monitors the water temperature, while the BMP sensor records ambient air pressure and temperature to consider external environmental factors.

User interaction is achieved through two capacitive touch sensors, allowing users to increase or decrease the target temperature easily. Based on this input and the current sensor data, the ESP32 activates or deactivates the relay, which in turn controls the heating coil in the electric kettle.

In addition to the OLED display, which shows real-time readings and system status locally, the project includes a web page interface hosted by the ESP32. This webpage can be accessed via any device—smartphone, tablet, or laptop—connected to the same network or remotely (if configured). It allows users to monitor real-time temperature, humidity, and system status wirelessly, adding a smart home and IoT dimension to the project. This makes the system convenient, accessible, and suitable for remote environments or user supervision without physical presence

In this mini project, we developed a smart water kettle system using an ESP32 microcontroller, which connects to the internet via Wi-Fi and hosts a web page to monitor real-time parameters. The web page was created using HTML and CSS, and the code was written in Micro Python using the Thonny IDE.
The purpose of the web page is to allow users to remotely monitor and control the kettle. The ESP32 collects temperature and humidity data using the DS18B20 temperature sensor and BMP180/BME280 sensor for outdoor humidity and pressure. The ESP32 serves this data over Wi-Fi by hosting a small web server

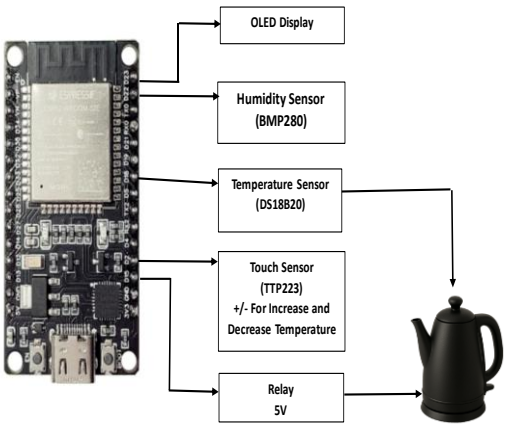


Fig.8 Process flow

.

The webpage displays the following key parameters:

- Water Temperature (°C) from DS18B20 sensor
- Outdoor Temperature and Humidity
- Preset Temperature Range (Min and Max)
- Kettle Status (ON/OFF)

Users can access this web page from mobile phones, laptops, or any internet-connected device, as long as they are connected to the same network or the ESP is configured in
station mode.

IV. Code for web page

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Smart Water Heater</title>
  <style>
    body {
      background-image:
url('https://i.imgur.com/jzdi8Tk.jpg'); /* You can
embed your own */
      background-size: cover;
      font-family: Arial, sans-serif;
      text-align: center;
      color: #333;
    }

    .container {
      margin: 100px auto;
      background-color: rgba(255, 255, 255, 0.9);
      padding: 30px;
      width: 400px;
      border-radius: 12px;
      box-shadow: 0 0 10px rgba(0,0,0,0.3);
    }

    h2 {
      margin-bottom: 5px;
    }

    .temp-box, .set-temp-box {
      margin: 20px 0;
    }

    input[type="range"] {
          width: 100%;
    }

    .status {
      padding: 15px;
      margin-top: 20px;
      font-weight: bold;
      font-size: 18px;
      border-radius: 8px;
    }

    .status.on {
      background-color: red;
      color: white;
    }

    .status.off {
      background-color: green;
      color: white;
    }

  </style>
 </head>
 <body>
  <div class="container">
  <h2>SMART WATER HEATER</h2>
    <p><strong>Condition:</strong> Overcast</p>

 <div class="temp-box">
  <p>Outdoor Temperature: <span id="outTemp">--
 </span>°C</p>
  <p>Pressure: <span id="pressure">--</span> hPa</p>
 </div>

 <div class="set-temp-box">
  <label for="setTemp">Set Temperature</label><br>
  <input type="range" id="setTemp" min="20" max="100"
value="60" oninput="updateLabel(this.value)">
  <p><span id="setLabel">60</span>°C</p>
  <button onclick="sendSetTemp()">Update</button>
 </div>

 <div id="heaterStatus" class="status off">Heating: OFF</div>
</div>

<script>
 function updateLabel(val) {
  document.getElementById('setLabel').innerText = val;
 }

 function fetchData() {
  fetch('/data')
   .then(response => response.json())
          .then(data => {
            document.getElementById('outTemp').innerText=
          data.temp;
            document.getElementById('pressure').innerText=
          data.pressure;
            document.getElementById('setTemp').value=
          data.set_temp;
            document.getElementById('setLabel').innerText=
          data.set_temp;

            letheater= document.getElementById('heaterStatus');
```

```
        if(parseFloat(data.temp)<
    parseFloat(data.set_temp)) {
                heater.innerText = "Heating: ON";
          heater.className = "status on";
        } else {
          heater.innerText = "Heating: OFF";
          heater.className = "status off";
        }
       });
    }

    function sendSetTemp() {
      letnewTemp=
parseFloat(document.getElementById('setTemp').
value);
      fetch('/set_temp', {
        method: 'POST',
        headers:                          {'Content-Type':
'application/json'},
        body: JSON.stringify({ set_temp: newTemp
})
      })
      .then(res => res.json())
      .then(() => fetchData());
    }

    setInterval(fetchData, 3000);
    fetchData();
  </script>
</body>
</html>
```

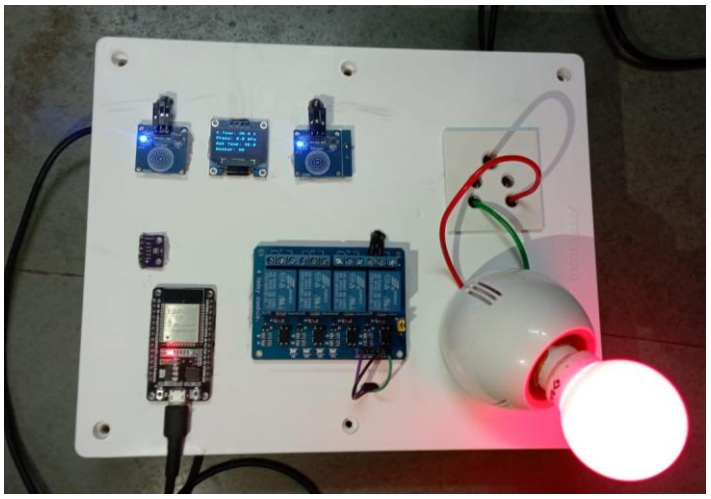
V. Code for interfacing

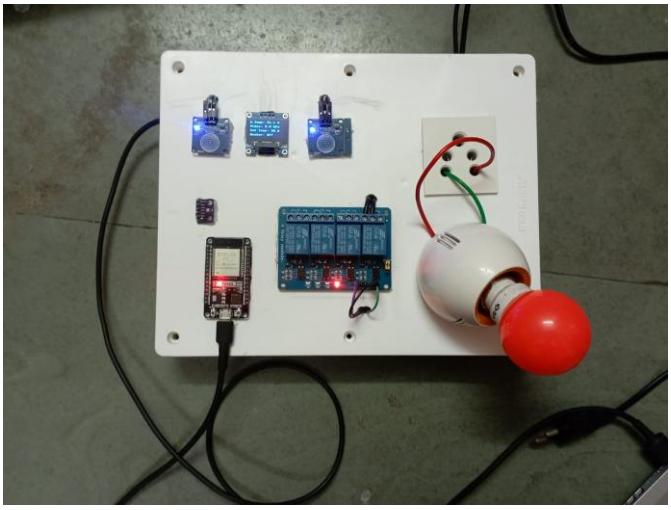
```
import usocket as socket
import network
from machine import Pin, SoftI2C, I2C
import machine
from time import sleep
import ujson
import _thread
import onewire
import ds18x20
import ssd1306
from bmp280 import BMP280

# ========== Wi-Fi Access Point Setup
==========
ssid = "KUNAL_ESP"
password = "kunal007"

ap = network.WLAN(network.AP_IF)
ap.active(True)
ap.config(essid=ssid, password=password,
authmode=network.AUTH_WPA_WPA2_PSK)
while not ap.active():
pass

ip = ap.ifconfig()[0]
print("Access Point active. Connect to Wi-Fi:")
print("SSID:", ssid)
print("Access at: http://{}/".format(ip))

# ========== Sensor & Display Initialization
==========

# BMP280 on I2C(1)
i2c_bmp = I2C(1, scl=Pin(22), sda=Pin(21), freq=100000)
bmp = BMP280(i2c_bmp, addr=0x76)

# DS18B20 on GPIO4
ds_pin = Pin(4)
ow = onewire.OneWire(ds_pin)
ds = ds18x20.DS18X20(ow)
roms = ds.scan()
print("Found DS18B20 devices:", roms)

# OLED on SoftI2C (GPIO19=SCL, GPIO18=SDA)
i2c_oled = SoftI2C(scl=Pin(19), sda=Pin(18))
oled = ssd1306.SSD1306_I2C(128, 64, i2c_oled)

# ======= Startup OLED Message =======
oled.fill(0)
oled.text("SMART WATER", 20, 0)
oled.text("HEATER", 40, 10)
oled.text("BY: KUNAL PATIL", 0, 30)
oled.text("    TANUJA PATIL", 0, 45)
oled.show()
sleep(3)
oled.fill(0)
oled.show()

# Touch sensors
touch_increase = Pin(33, Pin.IN)
touch_decrease = Pin(25, Pin.IN)

# GPIO output for relay
relay = Pin(15, Pin.OUT)

# Initial parameters
set_temp = 25
humidity = 40.0  # placeholder

# ========== Sensor Reading Function
==========
def read_bmp280():
try:
temperature, pressure =
bmp.read_compensated_data()
return round(temperature, 2), round(pressure / 100.0,
2)
except Exception as e:
print("BMP280 Error:", e)
```

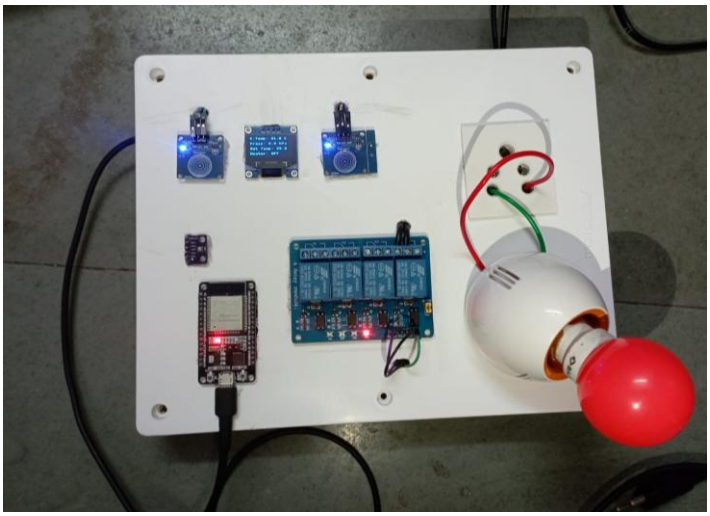
```
    return None, None

# ========== Web Server Handlers ==========
def load_html(filename):
    try:
        with open(filename, 'r') as file:
            return file.read()
    except:
        return "<h1>404 - File Not Found</h1>"

def handle_web_request(conn, request):
    global set_temp
    if 'POST /set_temp' in request:
        try:
            payload = request.split('\r\n\r\n', 1)[1]
            data = ujson.loads(payload)
            set_temp = float(data.get('set_temp', set_temp))
            print("Set temperature updated to:", set_temp)
            response = ujson.dumps({'success': True})
        except:
            response = ujson.dumps({'success': False})
        conn.send('HTTP/1.1 200 OK\nContent-Type: application/json\n\n')
        conn.sendall(response.encode())

    elif 'GET /data' in request:
        temp_bmp, pressure = read_bmp280()
        if temp_bmp is not None:
            response = ujson.dumps({'temp': temp_bmp, 'pressure': pressure, 'humidity': humidity})
        else:
            response = ujson.dumps({'temp': 'N/A', 'pressure': 'N/A', 'humidity': 'N/A'})
        conn.send('HTTP/1.1 200 OK\nContent-Type: application/json\n\n')
        conn.sendall(response.encode())

    elif 'GET /Next.html' in request:
        response = load_html('Next.html')
        conn.send('HTTP/1.1 200 OK\nContent-Type: text/html\n\n')
        conn.sendall(response)

    else:
        response = load_html('index.html')
        conn.send('HTTP/1.1 200 OK\nContent-Type: text/html\n\n')
        conn.sendall(response)

    conn.close()

def web_server():
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind(('', 80))
    s.listen(5)
    print("Web server started at http://{}".format(ip))
    while True:
        conn, addr = s.accept()
        print("Request from:", addr)
        request = conn.recv(1024).decode()
        handle_web_request(conn, request)

# ========== Control Logic ==========
def control_loop():
    global set_temp
    if not roms:
        print("No DS18B20 sensor found.")
        return

    while True:
        ds.convert_temp()
        sleep(0.75)

        for rom in roms:
            temp_ds = ds.read_temp(rom)
        temp_bmp, pressure = read_bmp280()

        # Touch inputs
        if touch_increase.value():
            set_temp = min(set_temp + 1, 100)
            print("[Touch] Increase button pressed. Set Temp:", set_temp)
            sleep(0.2)

        if touch_decrease.value():
            set_temp = max(set_temp - 1, 1)
            print("[Touch] Decrease button pressed. Set Temp:", set_temp)
            sleep(0.2)

        # Heater control logic
        if temp_ds < set_temp - 1:
            relay.value(1)
            relay_status = "ON"
        elif temp_ds > set_temp + 1:
            relay.value(0)
            relay_status = "OFF"
        else:
            relay_status = "IDLE"

        # OLED Display Output
        oled.fill(0)
        oled.text("C.Temp: {:.1f} C".format(temp_ds), 0, 0)
        if pressure is not None:
            oled.text("Press: {:.1f} hPa".format(pressure), 0, 16)
        else:
            oled.text("BMP280 Err", 0, 16)
        oled.text("Set Temp: {:.1f}".format(set_temp), 0, 32)
        oled.text("Heater: {}".format(relay_status), 0, 48)
        oled.show()

        # Thonny Shell Output
        print("\n===== STATUS =====")
        print("Current Temp   : {:.2f} °C".format(temp_ds))
        if pressure is not None:
```

```
            print("BMP280 Pressure: {:.2f}
        hPa".format(pressure))
                    else:
                print("BMP280 Error")
            print("Set Temp      : {:.2f}
        °C".format(set_temp))
    print("Heater Status  : {}".format(relay_status))
        print("====================")

                    sleep(0.5)

# ========== Main ==========
                    def main():
    _thread.start_new_thread(web_server, ())
                    control_loop()

                    main()
```

VI. Performance Analysis







VI. Result and Conclusion

I.Advantages

- **Remote Monitoring**: Users can view live data from the kettle on their smartphones, laptops, or any device with a browser.
- **Cross-Device Compatibility**: The web interface works across different platforms (Android, iOS, Windows, etc.).
- **No App Needed**: Simple browser access eliminates the need for installing external apps.
- **Smart Home Integration Ready**: The web UI opens doors for integration with home automation ecosystems.

II. Application

- **Home Automation**: Smart kettles for kitchens, dorm rooms, or compact living spaces.
- **Laboratories**: Controlled heating for water baths or chemical solutions
- **Smart Coffee/Tea Makers**: Integrate into automated beverage machines for temperature-based brewing.

III.future Scope

- **Mobile App Integration:**
  A dedicated mobile application can be developed to control and monitor the kettle from anywhere. Users can set temperature, get alerts, and view real-time status through their phone.
- **Voice Assistant Support:**
  The smart kettle can be connected with voice assistants like **Amazon Alexa** or **Google Assistant** for hands-free operation.

IV. References

- Wikipedia
- https://www.electronicwings.com/esp32/introduction-to-esp32
- https://en.wikipedia.org/wiki/Relay#:~:text=Relays%20are%20used%20to%20control,incoming%20signal%20onto%20another%20circuit.
- https://www.instructables.com/Cayenne-IoT-Geyser/
- https://www.silabs.com/developer-tools/usb-to-uart-bridge-vcp-drivers