REPORT

ON

FOUR WEEKS OF INTERNSHIP

Carried out on

Smart Mug Project

Submitted to

by

ANAGHA RAO

USN **4NM21IS016**

Under the guidance of

Vijay GH

Head of Technology

Dlithe

ACKNOWLEDGEMENT

Our internship would not have been a success without the encouragement, direction, and support of a lot of different people.

I would like to express my gratitude to my mentor, Mr. Vijay, for his guidance, support, and invaluable insights throughout the duration of this project. His expertise and willingness to share his knowledge have been invaluable, and I am grateful for his contributions to my learning.

I would also like to extend my thanks to DLithe, the company that provided me with the opportunity to work on this project. The experience and skills that I have gained from working with DLithe will be instrumental in my future endeavors.

I would like to thank my teammates for their collaboration, hard work, and dedication to this project. Their contributions have been critical to the success of our team, and I am grateful for the opportunity to work alongside them.

Finally, we would like to thank all our friends and well-wishers who have helped us whenever needed and supported us.

Anagha Rao

4NM21IS016

Information Science Department

TABLE OF CONTENTS

Abstract

The Smart Mug project is an innovative device that is designed to maintain the temperature of a beverage at a specified level. It is equipped with a temperature sensor, PTC heater and an Arduino microcontroller, which is programmed to regulate the heating elements inside the mug. The device features a local web interface, which has been developed using the HTML, CSS, and JavaScript languages.

Firstly, it saves time and energy as it eliminates the need for reheating beverages. Secondly, it ensures that the drink is always at the desired temperature. The mug is also user-friendly and easy to operate, with a simple interface.

The implementation of this project involves several steps, which included the assembly of the hardware components, the programming of the Arduino microcontroller, and the development of the web interface. The result is a functional and reliable prototype device that offers a practical solution to the problem of maintaining beverage temperature.

Overall, the Smart Mug project is a significant achievement in the field of temperature regulation and offers a glimpse into the future of smart devices

CHAPTER 1: Introduction

The Internet of Things (IoT) is a rapidly growing field that is revolutionizing the way we interact with technology. IoT refers to the network of devices that are connected to the internet, enabling them to exchange data and interact with each other. The potential applications of IoT are vast and varied, ranging from smart homes and cities to healthcare and transportation.

IoT devices are characterized by their ability to collect and transmit data, as well as their ability to be controlled remotely. These devices are equipped with sensors, processors, and communication modules that allow them to monitor and respond to changes in their environment. The data collected by IoT devices can be analysed and used to improve processes, optimize resource utilization, and enhance user experiences.

The Smart Mug project is a prime example of an IoT application, as it combines hardware and software components to create a connected device that can be controlled. The Smart Mug project is a temperature regulating mug that is designed to keep your beverage at your specified temperature. The project highlights the potential of IoT to create innovative solutions to everyday problems.

The device is equipped with a temperature sensor and an Arduino microcontroller, which work together to regulate the heating elements inside the mug. The mug also has a web interface that allows you to control the temperature of your drink. The web interface is built using HTML, CSS, and JavaScript.

Traditional mugs require regular reheating, which can be time and energy consuming. Additionally, it can be difficult to maintain a drinking temperature, especially when you are on the go. The Smart Mug project solves these problems by providing a convenient and easy-to-use solution that ensures your beverage is always at the desired temperature.

This report provides a detailed overview of the Smart Mug project. The report begins with a literature review that highlights the existing research on temperature regulation and smart devices. The literature review provides the context for the Smart Mug project and identifies the advantages of the device.

The report then moves on to describe the design and implementation of the Smart Mug project, including the hardware and software components. The report discusses the programming of the Arduino microcontroller, the development of the web interface, and the integration of the various components.

The testing and results section of the report describes the experimental setup used to evaluate the performance of the Smart Mug. The section provides details on the feedback obtained during the testing process.

The report concludes with a discussion of the strengths and weaknesses of the Smart Mug project, including its limitations and potential for future development.

CHAPTER 2: Literature Review

Temperature regulation has been a topic of interest for many years. The ability to control temperature is critical in many industries, including food and beverage, healthcare, and electronics. In recent years, the development of smart devices has opened new possibilities for temperature regulation, with devices such as smart thermostats and refrigerators becoming common.

The Smart Mug project builds on this trend by offering a solution to the problem of maintaining beverage temperature. The device uses a combination of hardware and software components to regulate the temperature of the beverage inside the mug. The device is user-friendly and easy to operate, with a simple and intuitive interface. This literature review aims to examine the existing research on smart mugs and explore the potential benefits and limitations of the Smart Mug project.

2.1 Temperature Regulation:

Temperature regulation is a complex process that involves the measurement and control of temperature in each system. Several technologies have been developed for temperature regulation, including refrigeration, heating, and cooling systems, and thermoelectric devices. These technologies vary in their efficiency, cost, and complexity, depending on the specific application.

2.2 Smart Devices:

Smart devices are a relatively new development that has gained popularity in recent years. These devices are designed to be connected to the internet and to interact with other devices and users. Smart devices are characterized by their ability to collect and analyse data, respond to user input, and adapt to changing environments.

Smart devices are increasingly being used for temperature regulation, with smart thermostats and refrigerators becoming more common. These devices offer several advantages over traditional systems, including increased efficiency, ease of use, and remote-control capabilities.

2.3 Benefits of Smart Mugs:

The use of smart mugs can reduce energy consumption and save time compared to traditional mugs. Smart mugs could maintain the temperature of the beverage at the desired level for an extended period, reducing the need for reheating.

2.4 Limitations of Smart Mugs:

Despite their potential benefits, smart mugs also have some limitations. One limitation is the cost of the device, which can be higher than that of traditional mugs. Additionally, smart mugs require a power source, which may not be readily available in some settings.

CHAPTER 3: Design and Implementation

3.1 Hardware Components

3.1.1 Apparatus

1. DLithe iot cube (Arduino Adafruit Metro ESP32)
2. DHT11 sensor
3. Wires
4. PTC Heater
5. 12V adaptor

3.1.2 Apparatus specifications

1. DHT11 sensor

   Digital temperature and humidity sensors like the DHT11 are inexpensive. Any micro-controller, including Arduino, Raspberry Pi, etc., may easily interface with this sensor to instantly monitor humidity and temperature.

   The DHT11 sensor is made up of a capacitive humidity measuring element and a thermistor for temperature detection. A moisture-holding substrate serves as a dielectric between the two electrodes of the humidity sensor capacitor. The capacitance value changes as the humidity level changes. The IC measures, processes, and converts the resistance values into digital form.

   DHT11 has a temperature range of 0 to 50 degrees Celsius with a 2-degree accuracy. This sensor has a humidity range of 20 to 80% with a 5% accuracy. This sensor has a sampling rate of 1Hz, which means it takes one reading per second. DHT11 is a tiny device with an operational voltage range of 3 to 5 volts.

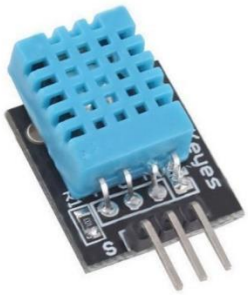

Figure 1.1: DHT11 sensor 1

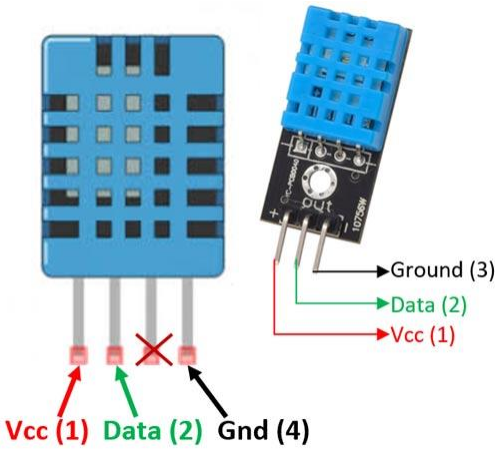

Figure 1.2: DHT11 Sensor Pinout 1

2. PTC heater

12V PTC (Positive Temperature Coefficient) Heaters is used for heating air. Made up of an Aluminum shell, the PTC material has the characteristic that it is the constant temperature on the surface can be dry heated for a long time.

Applications:

- Heater for water, cup, gas,
- A hair straightener, yogurt maker, Instrumentation
- dehumidification, Chocolate extrusion, coffee maker.
- Car and components preheating

Features:

- 12V Constant Temperature PTC Heating Element Thermostat Heater Plate.
- Widely used in foot bath devices, water boiler, yogurt maker, chocolate extrusion, coffee maker.
- Made of Aluminum shell, eco-friendly, durable, and sturdy to use.
- Constant temperature and surface insulation, safe to use.
- Also used for car and components preheating.
- Can be dry heated for a long time

Specifications:

- Heating Material: PTC Thermistor
- Shell Material: Aluminum
- Lead wire: Silicone Line
- Voltage: 12V
- Power: 30W
- Surface Dry Heating Temperature: 220ºC
- Dimension: 35 x 21 x 5mm (Approx.)

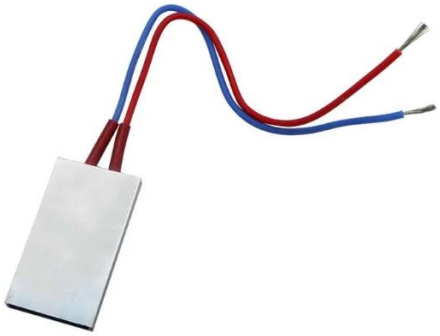

Figure 2: PTC heater 1

3.1.3 The Circuit Connections

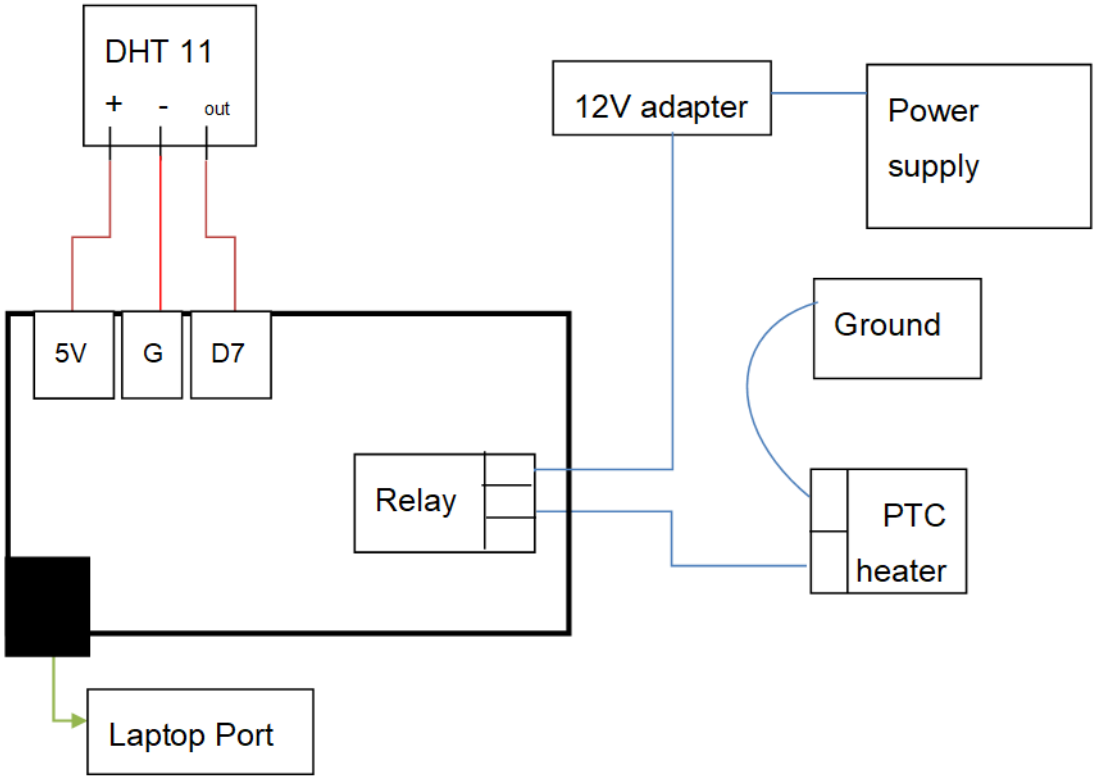

Figure 3: Circuit Connections 1

The above diagram shows how to connect the different hardware components.

1. DHT11 sensor

   To connect the sensor to the IoT cube, we must first attach the Vcc '+' pin of the sensor to the 5V supply pin on the cube. Next, connect the ground '-' pin of the sensor to the G pin, which is the ground pin on the IoT cube. Finally, attach the out pin of the sensor to digital pin 7, also known as the D7 pin, on the IoT cube.

   This will enable the out pin to send the data it reads from the sensor to the D7 pin, which we can use to access the data via the serial monitor after running the code.

2. PTC heater

   To properly connect the PTC heater to the relay and microcontroller, there are a few steps to follow. First, connect the positive lead of the 12V

adapter to the common (COM) pin of the relay. This will provide power to the relay.

Next, connect one wire of the PTC heater to the NO (normally open) pin of the relay. The NO pin is where the power will flow through to the PTC heater when the relay is activated. Finally, connect the other wire of the PTC heater to the ground. This will complete the circuit and allow the PTC heater to be controlled by the microcontroller

To control the relay, connect the control input of the relay to one of the GPIO pins of the microcontroller. Here, we will use digital pin 5 or D5. Once the wiring is complete, write a program for the microcontroller to control the relay based on the temperature reading from the PTC heater

3.2 Software Components

The program was written in an Arduino IDE version 1.8.19 using C language. This code is of an ESP32 program written using the Arduino IDE to control a web-based temperature control system. It includes libraries such as WiFi, AsyncTCP, ESPAsyncWebServer, and DHT to enable communication with the ESP32 chip and other peripherals such as the DHT11 temperature and humidity sensor.

The code starts by defining some constants and variables such as DHTPIN and DHTTYPE for the DHT sensor and the heater variable for the relay that controls the mug temperature. The network credentials, SSID, and password are also defined in this section.

The code then creates an AsyncWebServer object and defines the HTML and JavaScript for the web-based interface. The interface includes a slider that sets the temperature and a function that sends the set temperature to the server. There is also a string processor that replaces the slider value with a dynamic value.

In the setup function, the serial port is initiated for debugging purposes. The GPIO pins for the relay and DHT sensor are defined as output and input, respectively. The PWM properties are set up, and the channel is attached to the GPIO to be controlled. PWM stands for Pulse-Width Modulation, which is a technique used to control the power supplied to a device.

Attaching the PWM channel to a GPIO (General Purpose Input/Output) means that the GPIO pin on the microcontroller is configured to send the PWM

signal to the PTC heater. This allows the microcontroller to control the heat of the PTC heater by varying the duty cycle of the PWM signal, without having to change the power supply voltage directly.The Wi-Fi is then connected to the ESP32 chip.

The root web page is defined, and the server is started. In the loop function, the temperature and humidity readings are taken from the DHT11 sensor and printed on the serial monitor. The set temperature value is also checked, and if it is greater than or equal to one, the heater is turned on.

The required libraries were downloaded and are as follows:

```
// Import required libraries
#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include "DHT.h"
```

Then define the required input pins and sensors and define the DHT pin and type

```
#define DHTPIN 7     //pin of sensor
#define DHTTYPE DHT11
int heater = 5;//pin of relay already inbuilt
DHT dht(DHTPIN, DHTTYPE);
```

This section of the code declares some variables and constants and sets up an AsyncWebServer object on port 80. The frequency, channel, and resolution of the Pulse Width Modulation (PWM) output are being defined in the code. PWM is a technique used to simulate an analog voltage output using a digital signal. By varying the duty cycle of the PWM signal, the effective voltage output can be changed. The frequency of the PWM signal determines how often the output voltage is updated, and the resolution determines how many discrete voltage levels can be produced. Therefore, by setting these properties, the code is defining how the heater attached to the PWM output will be controlled.

```
// Replace with your network credentials
```

```
const char* ssid = "Mug";//name of the network
const char* password = "hi22me03";//network password
const int output = 42;
String sliderValue = "0";
// setting PWM properties
const int freq = 5000;
const int ledChannel = 0;
const int resolution = 8;
const char* PARAM_INPUT = "value";
// Create AsyncWebServer object on port 80
AsyncWebServer server(80);
```

This section of code defines a string variable named index html that contains an HTML document. The PROGMEM keyword specifies that the string is stored in the program memory of the microcontroller instead of the RAM.

The HTML document contains a slider input that is used to select the desired temperature. The value of the slider is displayed using the span element with the id textSliderValue. The onchange event of the slider is used to trigger the updateSliderPWM function, which sends an HTTP GET request to the server with the value of the slider as a parameter.

The updateSliderPWM function retrieves the value of the slider and updates the text of the span element. It then creates an XMLHttpRequest object and sends an HTTP GET request to the server with the value of the slider as a parameter.

```
const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html>
<head>
```

```
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>ESP Web Server</title>
  <style>
    html {font-family: Arial; display: inline-block; text-align: center;}
    h2 {font-size: 2.3rem;}
    p {font-size: 1.9rem;}
    body {max-width: 400px; margin:0px auto; padding-bottom: 25px;}
    .slider { -webkit-appearance: none; margin: 14px; width: 360px; height: 25px; background: #FFD65C;
      outline: none; -webkit-transition: .2s; transition: opacity .2s;}
    .slider::-webkit-slider-thumb {-webkit-appearance: none; appearance: none; width: 35px; height: 35px; background: #003249; cursor: pointer;}
    .slider::-moz-range-thumb { width: 35px; height: 35px; background: #003249; cursor: pointer; }
  </style>
</head>
<body>
  <h2>Select Temperature</h2>
  <p><span id="textSliderValue">%SLIDERVALUE%</span></p>
  <p><input type="range" onchange="updateSliderPWM(this)" id="pwmSlider" min="0" max="80" value="%SLIDERVALUE%" step="1" class="slider"></p>
<script>
function updateSliderPWM(element) {
```

```
  var sliderValue =
document.getElementById("pwmSlider").value;
  document.getElementById("textSliderValue").innerHTML =
sliderValue;
  console.log(sliderValue);
  var xhr = new XMLHttpRequest();
  xhr.open("GET", "/slider?value="+sliderValue, true);
  xhr.send();
}
</script>
</body>
</html>
)rawliteral";
```

This is a function called processor() that replaces a placeholder in the HTML page with the value of a variable called sliderValue.

In the HTML page, there is a placeholder %SLIDERVALUE% that needs to be replaced with the actual value of sliderValue. This function takes the placeholder as input and checks if it matches with the string "SLIDERVALUE". If it does, it returns the value of sliderValue which replaces the placeholder in the HTML page. If it doesn't match, it returns an empty string.

This function is used in the server.on() function to set up a request handler that replaces the placeholder with the actual value of sliderValue whenever a client requests the web page.

```
// Replaces placeholder with button section in your web page
String processor(const String& var){
  //Serial.println(var);
  if (var == "SLIDERVALUE"){
    return sliderValue;
```

```
  }
  return String();
}
```

This setup() function is a part of an Arduino sketch that sets up the ESP32 board to run a web server and control PTC heater via PWM (Pulse Width Modulation).Here are the actions that take place inside the setup() function:

- Serial.begin(115200) initializes a serial connection between the ESP32 and a computer for debugging purposes. The baud rate is set to 115200.
- Serial.println(F("DHT11")) prints "DHT11" to the serial monitor.
- pinMode(heater, OUTPUT) sets the pin labeled heater as an output.
- pinMode(DHTPIN, INPUT) sets the pin labeled DHTPIN as an input.
- dht.begin() initializes communication with the DHT11 temperature and humidity sensor.
- ledcSetup(ledChannel, freq, resolution) configures the PWM properties for the heater with the specified ledChannel, freq, and resolution values.
- ledcAttachPin(output, ledChannel) attaches the output pin to the PWM channel that was set up in the previous step.
- ledcWrite(ledChannel, sliderValue.toInt()) sets the initial PWM duty cycle value for the PTC heater using the sliderValue variable.
- WiFi.begin(ssid, password) connects to a Wi-Fi network using the SSID and password variables.
- while (WiFi.status() != WL_CONNECTED) waits for the ESP32 to connect to the Wi-Fi network.
- Serial.println(WiFi.localIP()) prints the IP address of the ESP32 to the serial monitor.
- server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){...}) creates a route for the root URL ("/") and sends the index_html file as a response to any HTTP GET requests received on that route.
- server.on("/slider", HTTP_GET, [] (AsyncWebServerRequest *request) {...}) creates a route for "/slider" and listens for HTTP GET requests. When a request is received, it extracts the value of the value parameter from the

URL, sets the sliderValue variable to that value, and sets the PWM duty cycle of the PTC heater to the new value.

- server.begin() starts the web server and begins listening for incoming HTTP requests.

```
void setup(){
  // Serial port for debugging purposes
  Serial.begin(115200);
  Serial.println(F("DHT11"));
pinMode(heater, OUTPUT);//defines relay as output
pinMode(DHTPIN, INPUT);//defines dht sensor as input
  dht.begin();
  // configure PWM functionalitites
  ledcSetup(ledChannel, freq, resolution);
  // attach the channel to the GPIO to be controlled
  ledcAttachPin(output, ledChannel);
  ledcWrite(ledChannel, sliderValue.toInt());
  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi..");
  }
  // Print ESP Local IP Address
  Serial.println(WiFi.localIP());
  // Route for root / web page
```

```
  server.on("/", HTTP_GET, [](AsyncWebServerRequest
*request){
    request->send_P(200, "text/html", index_html,
processor);
  });
  // Send a GET request to
<ESP_IP>/slider?value=<inputMessage>
  server.on("/slider", HTTP_GET, [] (AsyncWebServerRequest
*request) {
    String inputMessage;
    // GET input1 value on
<ESP_IP>/slider?value=<inputMessage>
    if (request->hasParam(PARAM_INPUT)) {
      inputMessage = request->getParam(PARAM_INPUT)-
>value();
      sliderValue = inputMessage;
      ledcWrite(ledChannel, sliderValue.toInt());
    }
    else {
      inputMessage = "No message sent";
    }
    Serial.println(inputMessage);
    request->send(200, "text/plain", "OK");
  });
  // Start server
  server.begin();
}
```

The loop() function is the main program loop that repeatedly runs after the setup function has executed. In this code, it first reads the temperature and humidity from the DHT sensor and checks if the readings are valid. If the readings are not valid, it prints an error message and returns from the function.

If the readings are valid, it prints the temperature and humidity to the serial monitor. Then, it waits for two seconds before proceeding to the next iteration of the loop.

The code then checks if the sliderValue variable is greater than or equal to 1. If it is, it turns on the heater by setting the heater pin to HIGH. It then checks if the temperature is greater than or equal to the sliderValue value. If it is, it turns off the heater by setting the heater pin to LOW.

In other words, the code uses the sliderValue variable, which is set by the user using a slider in the web interface, to control the heater. If the slider value is greater than or equal to 1, the heater is turned on and off based on the current temperature and the sliderValue value.

```
void loop() {
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();


  // Check if any reads failed and exit early (to try
again).
  if (isnan(temperature) || isnan(humidity)) {
    Serial.println(F("Failed to read from DHT sensor!"));
    return;
  }
//prints in serial monitor
  Serial.print(F("Humidity: "));
  Serial.print(humidity);
  Serial.print(F("%  Temperature: "));
  Serial.print(temperature);
```

```
  Serial.println(F("°C "));
  // Wait a few seconds between measurements.
  delay(2000);
//the logic for the mug
  if(sliderValue.toInt() >= 1){
    Serial.println(sliderValue.toInt());
    digitalWrite(heater, HIGH);
    if(temperature >= sliderValue.toInt())
    digitalWrite(heater, LOW);
  }
}
```

The code uses the HTTP (Hypertext Transfer Protocol) protocol to communicate between the web client (browser) and the ESP32 web server. The code sends an HTTP GET request to the server with the new slider value every time the slider is moved, and the server responds with an HTTP OK response.

CHAPTER 4: Testing and results

4.1 Testing

After connecting the circuit to the power supply and laptop, compile and upload the program to the IoT cube. Ensure that the correct connected port is displayed under the tools bar. Once the code is uploaded, open the serial monitor where a message stating "Connecting to WiFi..." will be displayed. If the WiFi network name and password are correct, an IP address will be generated as shown in Fig 4. Once the IP address is generated, the sensor data will be displayed in the serial monitor. Type the IP address into any browser connected to the same WiFi network as the program, and a webpage will be displayed as shown in Fig 5. Slide the slider button to set your desired temperature. This value will be returned to the program and displayed in the serial monitor as well. If the temperature set is higher than the current temperature being read by the DHT11 sensor, the relay will switch ON, and the heater will start heating the mug. Otherwise, the relay will remain OFF.

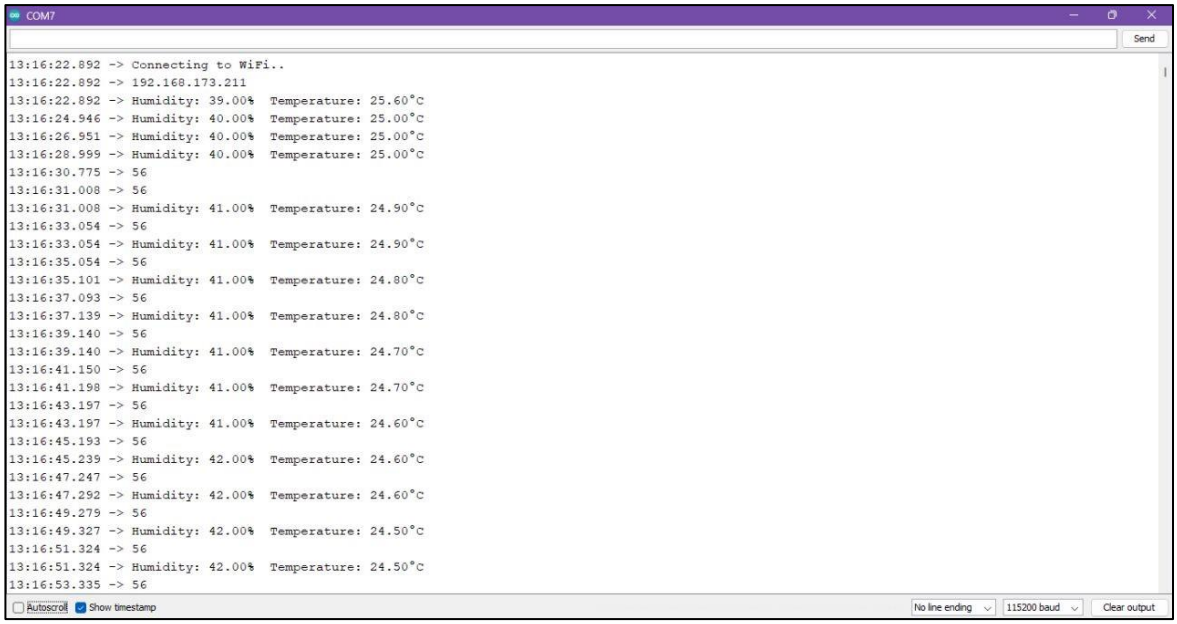

Figure 4: Serial Monitor 1

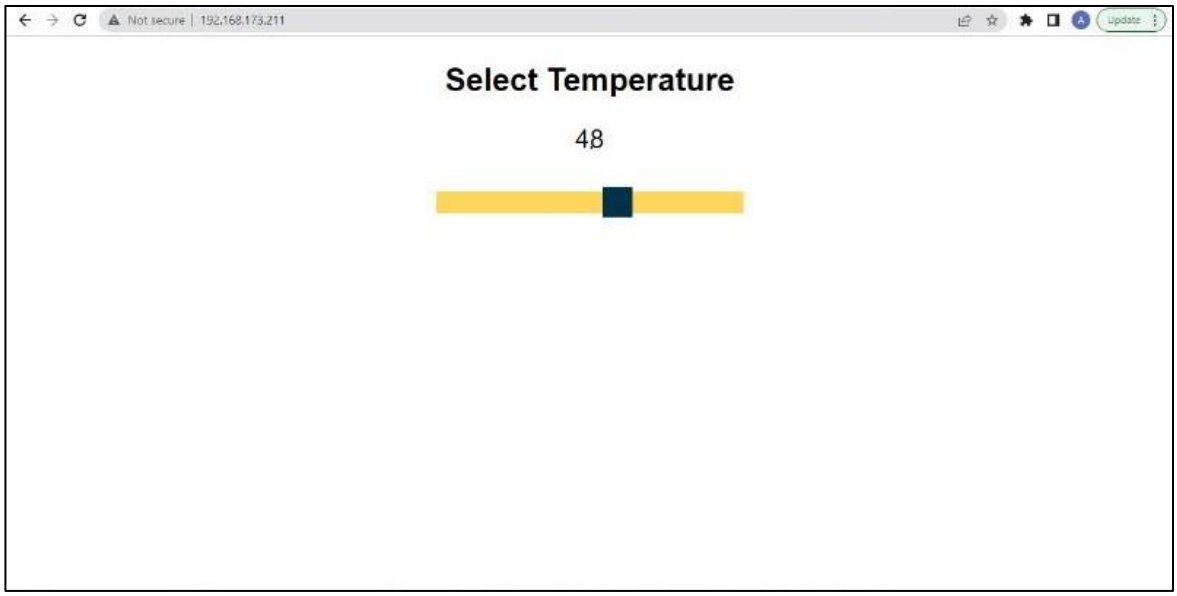

Figure 5: Web Page 1

4.2 Expected Result

After setting the temperature, it is expected that the PTC heater will start heating up immediately and reach the desired temperature within two to three minutes.

4.3 Result

The circuit and code were functioning properly. However, the PTC heater exhibited slow heating even with a 12 V adaptor. This may be attributed to inadequate current supply.

CHAPTER 5: Conclusion

Future enhancements include linking the data to the cloud and controlling the temperature from any location. Another improvement might include smaller and more efficient sensors and heaters, allowing the mug to be more portable. The smaller parts will also aid in enclosing the circuit, therefore protecting it from external harm and water.

The code presented shows how to use an ESP32 microcontroller to read data from a DHT11 sensor and control a relay to turn on/off a heating element. The code also includes a web server that can be used to control the temperature of the heating element remotely through a web page. The code uses the HTTP protocol to communicate between the web server and the ESP32. The user can adjust the temperature of the heating element using a slider on the web page, and the ESP32 will adjust the output voltage to the heating element accordingly.

Overall, this code provides a simple and effective way to control the temperature of a device using an ESP32 microcontroller and a DHT11 sensor.

The Smart Mug project offers a novel solution to the problem of maintaining beverage temperature, with several potential benefits over traditional mugs. However, the project also has some limitations, including the cost of the device and the need for a power source. Further research is needed to explore the potential benefits and limitations of smart mugs and to assess their practical applications in various settings, including healthcare and home use. Overall, the Smart Mug project represents a promising development in the field of smart devices and has the potential to revolutionize the way we enjoy our beverages.

Table of Figures

References

Anagha Rao (2023, May 7). Smart Mug Project [Internet of Things internship]. GitHub.

https://github.com/Ignyita/Smart-Mug.git