TABLE OF CONTENT

SMART WATER FOUNTAINS

ABSTRACT

Smart water fountains are a technological innovation designed to revolutionize the way people access and consume drinking water. This abstract provides an overview of the concept and benefits of smart water fountains. These innovative fountains incorporate sensors, data connectivity, and user-friendly interfaces to deliver clean, chilled, or even flavored water on demand. The integration of smart technology not only enhances user experience but also promotes sustainability by reducing single-use plastic bottle consumption and minimizing water wastage. This abstract explores the key features, environmental advantages, and potential applications of smart water fountains, emphasizing their role in promoting a healthier, more eco-conscious future.

Smart water fountains represent a technological evolution in the field of hydration and sustainability. This abstract provides an overview of the key concepts and benefits associated with these innovative devices. Smart water fountains integrate advanced sensor technology, IoT connectivity, and data analytics to provide users with convenient, safe, and eco-friendly access to drinking water. They offer features such as touchless operation, real-time water quality monitoring, and the ability to track personal hydration levels. Furthermore, these fountains contribute to a reduction in plastic waste by promoting the use of reusable containers. This abstract highlights the significance of smart water fountains in promoting healthier lifestyles and environmental conservation, making them a valuable addition to modern infrastructure.

Smart water fountains represent a promising advancement in the field of hydration management. This technology integrates IoT (Internet of Things) capabilities and sensor-driven data analysis to provide real-time information and support for individuals seeking to maintain optimal hydration levels. This abstract provides an overview of the key features and benefits of smart water fountains, highlighting their potential to revolutionize the way we approach hydration in various settings, from public spaces to workplaces and healthcare facilities.

Smart water fountains offer several innovative features. They are equipped with sensors that can detect user proximity, triggering water dispensing without physical contact, thereby promoting hygiene and reducing the spread of germs. Furthermore, these fountains provide customizable hydration plans based on individual needs, taking into account factors like age,

activity level, and climate conditions. Users can access this information through smartphone apps or integrated touchscreens, making it easy to track their daily water intake and set reminders for drinking water.

In addition to user-centric features, smart water fountains also enable facility managers to optimize water consumption, reduce waste, and monitor water quality. They can remotely monitor the status of fountains, detect leaks, and ensure water is filtered and chilled to the desired standards. This makes them a valuable addition to sustainable and eco-friendly initiatives in public and private spaces.

LIST OF FIGURE:

LIST OF ABBREVIATIONS:

ACRONYMS	EXPANSION
AWS	Amazon web service
RFID	Radio frequency identification
HTTP	Hyper text transfer protocol
MQTT	Message queuing telemetry transport

CHAPTER 1

INTRODUCTION

Welcome to the world of smart water fountains! In this presentation, we will explore how these innovative fountains are revolutionizing hydration. Discover the potential of these intelligent devices to improve access to clean drinking water and promote healthy lifestyles. Get ready to dive into the future of hydration!

1.1 Smart Water Fountain Features:

Smart water fountains come packed with exciting features such as real-time water quality monitoring, touchless operation, integrated hydration tracking, and customizable water preferences. These advancements ensure a safe, convenient, and personalized drinking experience for users.

1.2 Promoting Sustainability:

Smart water fountains play a crucial role in promoting sustainability. By encouraging the use of reusable bottles and reducing plastic waste, these fountains contribute to a greener environment. Additionally, they can be powered by renewable energy sources to minimize their carbon footprint.

1.3 Data-Driven Insights:

The data collected by smart water fountains provides valuable insights into hydration patterns and usage trends. Analyzing this data can help organizations and communities make informed decisions to improve water infrastructure, optimize maintenance, and promote healthier habits.

1.4 Enhancing Hydration Experience:

With smart water fountains, the hydration experience is taken to new heights. Users can enjoy customizable water temperature, flavor infusions, and even interactive games while hydrating. These features make drinking water more exciting, encouraging people to stay hydrated throughout the day.

CHAPTER 2

SYSTEM ANALYSIS:

2.1 EXISTING SYSTEM:

Planning and Assessment:

Identify the locations within your existing system where you want to install smart water fountains. Consider factors like accessibility, water supply, and power sources.

Selection of Smart Water Fountains:

Choose water fountains that have smart features, such as sensors for detecting users, water quality monitoring, and connectivity options (e.g., Wi-Fi or Bluetooth).

Connectivity:

Ensure that the smart water fountains can connect to your network. This may require setting up a dedicated Wi-Fi network or using other connectivity solutions.

Water Quality Monitoring:

Implement sensors to monitor water quality, ensuring that the water dispensed is safe and clean.

User Interaction:

Set up sensors or touch interfaces for users to activate the fountain. This could involve motion sensors, RFID cards, or smartphone apps.

Data Collection and Analysis:

Collect data from the smart fountains, including usage patterns, water quality data, and maintenance needs. Analyze this data to make informed decisions.

Maintenance and Alerts:

Implement a system for maintenance alerts. Smart fountains can generate alerts when filters need replacement or if there are any issues with the system.

Security:

Ensure that the smart fountains and the data they collect are secure. This involves network security and data encryption.

Integration with Existing Infrastructure:

If you have an existing water supply and plumbing system, make sure the smart fountains can seamlessly integrate with it.

User Education:

Educate users on how to use the smart water fountains and any additional features they offer.

Testing and Calibration:

Test the system thoroughly to ensure it functions as intended, including the sensors, water quality monitoring, and data collection**.**

Scaling and Expansion: If successful, consider scaling the smart water fountain system to other locations within your existing system.

2.1.1 DISADVANTAGE:

Water Quality Monitoring:

Smart fountains can continuously monitor water quality, ensuring that the water remains safe for consumption. If contamination is detected, alerts can be sent to authorities for immediate action.

Efficient Distribution:

These fountains can optimize water distribution, reducing wastage and ensuring a more even supply to different areas within the community.

Real-time Data:

Data from the smart fountains can be used for analysis and decision-making, helping to identify and address water supply issues promptly.

User-Friendly:

Features like touchless operation, bottle refill counters, and accessibility options can make the fountains more user-friendly, especially for people with disabilities.

Cost Savings:

Smart technology can help reduce operational costs by identifying and fixing leaks and other issues in the water supply system.

2.2 PROPOSED SYSTEM:

Water Quality Assurance:

Smart fountains can include water purification and filtration systems to ensure clean and safe drinking water.

Real-time Monitoring:

These fountains can continuously monitor water quality and consumption, providing data for immediate response to any issues.

Efficiency:

They can optimize water distribution, minimizing wastage and ensuring equitable access to water resources.

Maintenance Alerts:

Smart technology can detect and report issues like leaks or system malfunctions, allowing for timely maintenance and repairs.

User Convenience:

Features such as touchless operation, bottle refill counters, and accessible design can enhance user experience.

Data Analytics:

The data collected can be analyzed to make informed decisions about water resource management, infrastructure improvements, and resource allocation.

2.2.2 ADVANTAGES:

Water Quality Enhancement:

Smart fountains can provide high-quality, purified water, promoting health and well-being.

Convenience and Efficiency:

Users can easily access clean water with features like touchless operation, bottle refill counters, and efficient water distribution.

Sustainability:

Monitoring and data analytics can help ensure the responsible use of water resources, reducing waste and environmental impact.

Cost Savings:

Smart technology can lead to reduced operational costs by optimizing water distribution and promptly addressing maintenance issues.

CHAPTER 3

SYSTEM SPECIFICATION:

3.1 SOFTWARE SPECIFICATION:

- LINUX
- Programming Language python

3.2 HARDWARE SPECIFICATION:

- Filtration and Purification
- Capacity
- Sensors and flow rate
- Electrical connections and Solar panels
- Data transmission and Remote control
- Touchscreens and buttons
- RFID Readers
- ADA- compliant design

3.4 SOFTWARE DESCRIPTION:

PYTHON:

Python is a general-purpose language which means it is versatile and can be used to program many different types of functions. Because it is an interpreted language, it precludes the need for compiling code before execution and because it is a high-level programming language, Python is able to abstract details from code. In fact, Python focuses so much attention on abstraction that its code can be understood by most novice programmers.

Python code tends to be short and when compared to compiled languages like C and C++, it executes programs slower. Its user-friendliness makes it a popular language for citizen developers working with machine learning algorithms in low-code no-code (LCNC) software applications.

Python has a simply syntax and is known for having a large community that actively contributes to a growing selection of software modules and libraries. Python’s initial development was spearheaded by Guido van Rossum in the late 1980s. Today, Python is managed by the Python Software Foundation.

Techopedia Explains Python

Python offers several frameworks for web development. A Python Web framework is a group of modules and libraries that enable programmers to re-use another developer's code. This collaborative approach can developers avoid dealing with low-level issues such as protocols, sockets and process/thread management.

Python Frameworks

Here are 10 frameworks that web developers, machine learning teams and data analytics teams should consider when using Python:

Open-source Django is a popular Python web framework that facilitates quick web design and development. Django is a free-to-use framework that enables developers to reuse code to build high-quality web apps and APIs. Django is known for:

- Helping programmers avoid security blunders.
- Supporting a data-driven architecture.
- Moving software from concept to launch quickly.

Pyramid is a compact open-source web framework that works in all supported versions of Python. It offers the essential elements required for online applications including delivering static content and converting URLs to code. Some of Pyramid's attributes include:

- Security APIs that support authentication and authorization.
- A cookiecutter that generates sample Pyramid projects from project templates.

- Supporting the SQLAlchemy project and using its object-relational mapper (ORM) to interface with databases.

Bottle is a Web Server Gateway Interface (WSGI) micro-web framework for Python that is known for being lightweight and easy to use. Bottle is distributed as a single file module and the default Python library is the only dependency of the framework. It is is a popular framework for building mobile applications and supports:

- Python versions 2.7 and above.
- Mako, Jinja2, and Cheetah templates.
- WSGI-capable HTTP servers, including Bjoern, Google App Engine, fapws3 and CherryPy.
- URL mapping using condensed syntax.

CherryPy is an object-oriented HTTP framework that supports Apache and Microsoft IIS. Some of CherryPy's attributes include:

- A robust configuration system suitable for both developers and deployers.
- Built-in support for testing, coverage and profiling.
- Tools for authentication and caching.
- Flexible plugins.
- Robust configuration management.

Flask offers more control than its closest competitor, Django, and features support for unit testing. Along with RESTful request-dispatching and WSGI compatibility, Flask is known for:

- Providing an integrated development server with a debugger.
- Jinja2 templating (tags, filters, macros, and more).
- 100% compliance with WSGI 1.0.

Web2py allows developers to create, distribute, debug, test, manage a database and maintain applications. It has no setup files and can operate from a USB disk. Web2py can:

- Serve as a manual for web developers using the Model View Controller (MVC) paradigm.
- Automatically fix problems that may result in security risks.
- Support a database abstraction layer (DAL) that dynamically writes SQL is part of the framework.

Tornado is an open-source asynchronous framework for I/O operations. Tornado is known for supporting applications that require long-lived connections, real-time location services and allowing the integration of authentication and authorization methods from third parties.

BlueBream is a web application framework, server and library for Python programmers that was initially known as Zope 3. BlueBream is known for being durable, reliable and adaptive. It supports reusable software components as well as:

- WSGI (Web Server Gateway Interface) compatibility for Python.
- A template-development language that complies with XHTML.
- A program for creating forms automatically.

Grok

Grok is a robust framework for creating dependable and adaptable web applications. It supports DRY (Don't Repeat Yourself) software development and has a quick learning curve. Like other full-stack Python web frameworks, Grok features an intuitive UI (user interface).

Quixote

Quixote allows Python programmers to quickly create Web-based apps. This framework's objective is to offer web developers exceptional performance and flexibility for producing HTML with Python code

JSON

JSON, or JavaScript Object Notation, is a format used to represent data. It was introduced in the early 2000s as part of JavaScript and gradually expanded to become the most common medium for describing and exchanging text-based data. Today, JSON is the universal standard of data exchange. It is found in every area of programming, including front-end and server-side development, systems, middleware, and databases.

This article introduces you to JSON. You'll get an overview of the technology, find out how it compares to similar standards like XML, YAML, and CSV, and see examples of JSON in a variety of programs and use cases.

TABLE OF CONTENTS

A little bit of history

JSON was initially developed as a format for communicating between JavaScript clients and back-end servers. It quickly gained popularity as a human-readable format that front-end programmers could use to communicate with the back end using a terse, standardized format. Developers also discovered that JSON was very flexible: you could add, remove, and update fields ad hoc. (That flexibility came at the cost of safety, which was later addressed with the JSON schema.)

[Why Wasm is the future of cloud computing | The rise of WebAssembly]

In a curious turn, JSON was popularized by the AJAX revolution. Strange, given the emphasis on XML, but it was JSON that made AJAX really shine. Using REST as the convention for APIs and JSON as the medium for exchange proved a potent combination for balancing simplicity, flexibility, and consistence.

Next, JSON spread from front-end JavaScript to client-server communication, and from there to system config files, back-end languages, and all the way to databases. JSON even helped spur the

NoSQL movement that revolutionized data storage. It turned out that database administrators also enjoyed JSON's flexibility and ease of programming.

Today, document-oriented data stores like MongoDB provide an API that works with JSON-like data structures. In an interview in early 2022, MongoDB CTO Mark Porter noted that, from his perspective, JSON is still pushing the frontier of data. Not bad for a data format that started with a humble curly brace and a colon.

Why developers use JSON

No matter what type of program or use case they're working on, software developers need a way to describe and exchange data. This need is found in databases, business logic, user interfaces, and in all systems communication. There are many approaches to structuring data for exchange. The two broad camps are binary and text-based data. JSON is a text-based format, so it is readable by both people and machines.

JSON is a wildly successful way of formatting data for several reasons. First, it's native to JavaScript, and it's used inside of JavaScript programs as JSON literals. You can also use JSON with other programming languages, so it's useful for data exchange between heterogeneous systems. Finally, it is human readable. For a language data structure, JSON is an incredibly versatile tool. It is also fairly painless to use, especially when compared to other formats.

How JSON works

When you enter your username and password into a form on a web page, you are interacting with an object with two fields: username and passward.

CHAPTER 4

4.SYSTEM DESIGN:

4.1 ALGORITHM:

1. **Sensor Data Acquisition**:

   - Gather data from various sensors, including water level sensors, temperature sensors, motion sensors, and water quality sensors.

2. **Data Preprocessing**:

   - Filter and preprocess sensor data to eliminate noise and ensure data accuracy.

   - Convert analog sensor data to digital format for analysis.

3. **Data Analysis and Decision Making**:

   - Analyze the data to make decisions, such as:

     - Checking if the water level is below a certain threshold to prevent pump damage.

     - Monitoring water quality for contaminants.

     - Adjusting water temperature based on user preferences.

     - Detecting motion or proximity of users to activate the fountain.

4. **User Interaction**:

   - Implement user interfaces for controlling the fountain, such as mobile apps or web interfaces.

   - Allow users to set preferences like water temperature or fountain activation schedules.

5. **Control Mechanisms**:

- Control the fountain's pump, heating/cooling elements, and lighting based on the data analysis and user input.

- Ensure that the water level is maintained within safe limits.

6. **Security**:

- Implement robust security measures to protect the IoT system from unauthorized access and data breaches.

7. **Communication**:

- Establish communication protocols, like MQTT or HTTP, for transmitting data between the fountain and a central server or cloud platform.

8. **Remote Monitoring**:

- Enable remote monitoring and control of the water fountain through the internet.

- Allow users to receive alerts or notifications regarding any issues with the fountain.

9. **Energy Efficiency**:

- Implement power-saving mechanisms to conserve energy, such as scheduling the fountain's operation during specific hours.

10. **Data Storage**:

- Store historical data for analysis, troubleshooting, and future improvements.

11. **Maintenance and Self-Diagnostics**:

  - Implement self-diagnostic routines to detect and report faults in sensors or components.

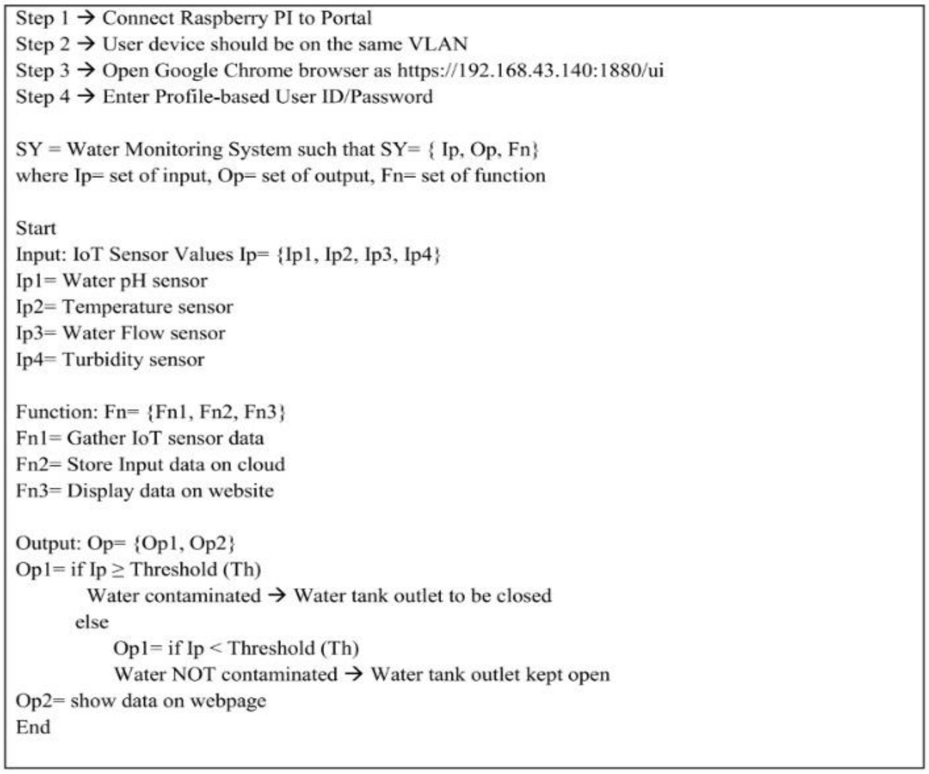
  - Provide maintenance alerts when the fountain requires cleaning, refilling, or other maintenance tasks.

12. **Machine Learning and Predictive Maintenance (Optional)**:

  - Employ machine learning algorithms to predict fountain maintenance needs based on historical data.

  - Implement predictive maintenance routines to reduce downtime.

13. **Scalability** - Design the system to be scalable, allowing for the addition of more fountains and sensors as needed.



```
Step 1 → Connect Raspberry PI to Portal
Step 2 → User device should be on the same VLAN
Step 3 → Open Google Chrome browser as https://192.168.43.140:1880/ui
Step 4 → Enter Profile-based User ID/Password

SY = Water Monitoring System such that SY= { Ip, Op, Fn}
where Ip= set of input, Op= set of output, Fn= set of function

Start
Input: IoT Sensor Values Ip= {Ip1, Ip2, Ip3, Ip4}
Ip1= Water pH sensor
Ip2= Temperature sensor
Ip3= Water Flow sensor
Ip4= Turbidity sensor

Function: Fn= {Fn1, Fn2, Fn3}
Fn1= Gather IoT sensor data
Fn2= Store Input data on cloud
Fn3= Display data on website

Output: Op= {Op1, Op2}
Op1= if Ip ≥ Threshold (Th)
        Water contaminated → Water tank outlet to be closed
      else
          Op1= if Ip < Threshold (Th)
          Water NOT contaminated → Water tank outlet kept open
Op2= show data on webpage
End
```

4.2 SYSTEM ARCHITECTURE:

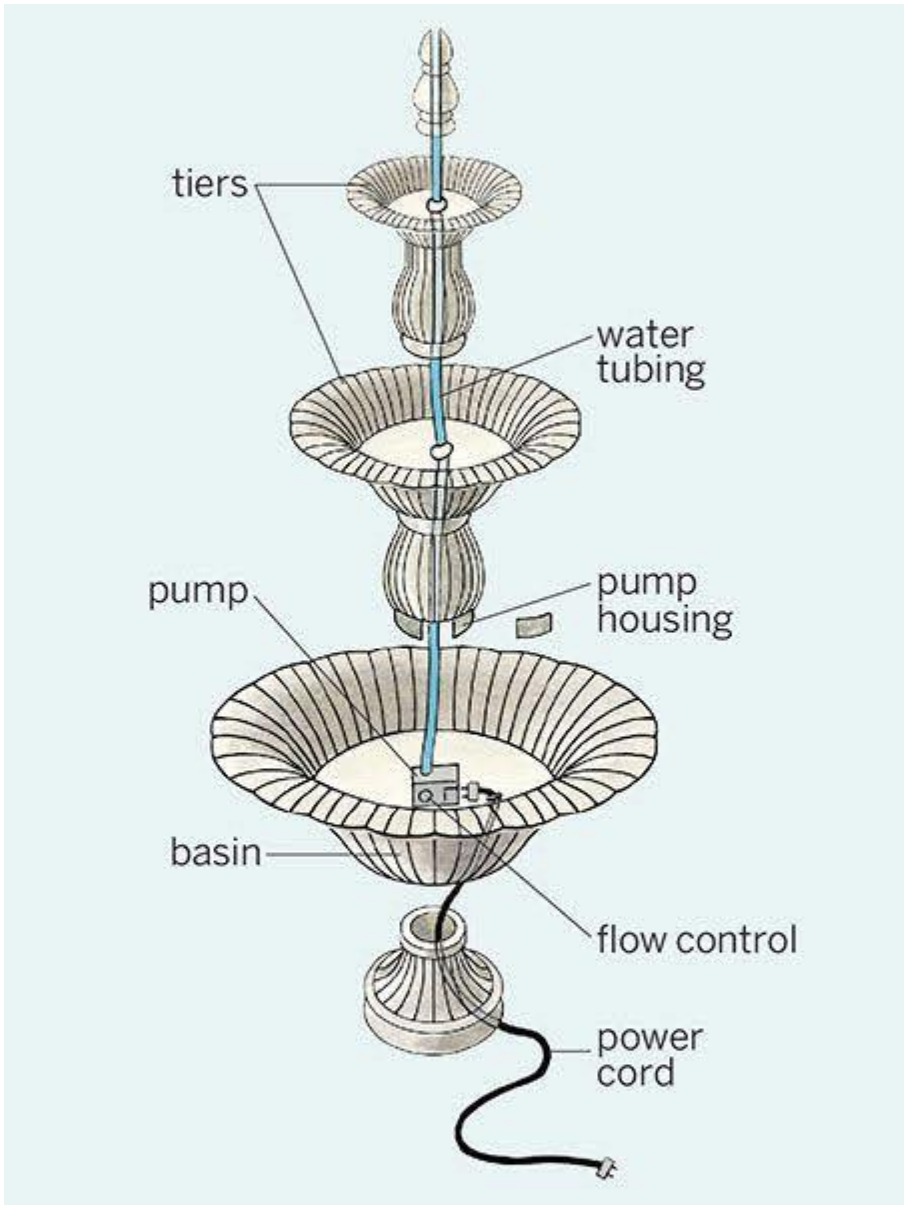


Fig 4.2 System architecture

4.3 CIRCUIT DIAGRAM :

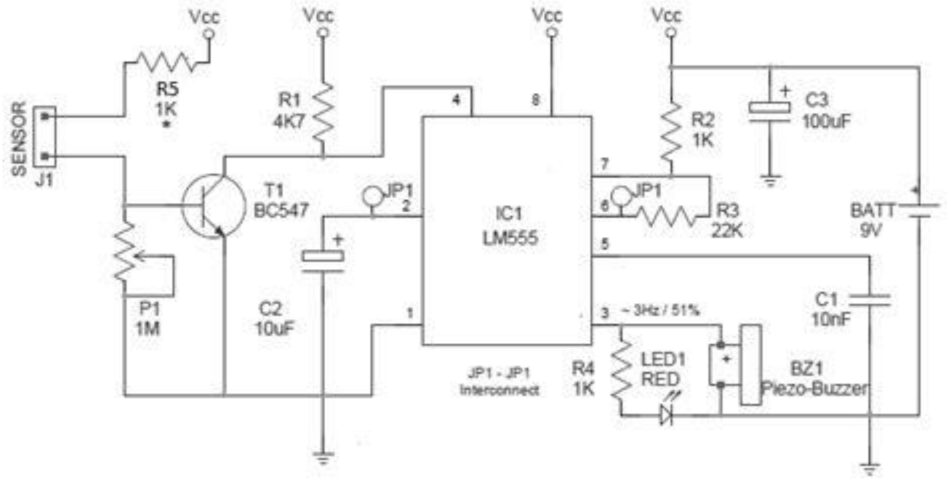


Fig 4.3 Circuit diagram

CHAPTER 5

SYSTEM IMPLEMENTATION

5.1 FRONT END SOURCE CODE

```
#define PIN_TRIG 27
#define PIN_ECHO 26
#define LED 18
#define MOTOR 27
unsigned int level=0;
#include <WiFi.h>
#include <PubSubClient.h>
#include <DHTesp.h>

const int DHT_PIN = 15;
DHTesp dht;
const char* ssid = "Wokwi-GUEST"; ///  wifi ssid
const char* password = "";
const char* mqtt_server = "test.mosquitto.org";// mosquitto server url

WiFiClient espClient;
PubSubClient client(espClient);
unsigned long lastMsg = 0;
float temp = 0;
float hum = 0;

void setup_wifi() {
  delay(10);
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  randomSeed(micros());

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}
void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }}
void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    String clientId = "ESP32Client-";
    clientId += String(random(0xffff), HEX);
    if (client.connect(clientId.c_str())) {
      Serial.println("Connected");
      client.publish("/ThinkIOT/Publish", "Welcome");
      client.subscribe("/ThinkIOT/Subscribe");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }}
}
```

```
void setup() {
  pinMode(LED,OUTPUT);
  digitalWrite(LED,HIGH);
  pinMode(MOTOR,OUTPUT);
  digitalWrite(MOTOR,LOW);
  Serial.begin(115200);
  pinMode(PIN_TRIG, OUTPUT);
  pinMode(PIN_ECHO, INPUT);
   pinMode(2, OUTPUT);
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
  dht.setup(DHT_PIN, DHTesp::DHT22);
}

void loop() {
  // Start a new measurement:
  digitalWrite(PIN_TRIG, HIGH);
  delayMicroseconds(10);
  digitalWrite(PIN_TRIG, LOW);

  // Read the result:
  int duration = pulseIn(PIN_ECHO, HIGH);
  Serial.print("Distance in CM: ");
  Serial.println(duration / 58);
  Serial.print("Distance in inches: ");
  Serial.println(duration / 148);

  level=duration/58;

  if(level<100)
  {
    digitalWrite(LED,HIGH);
  }
  else
  {
    digitalWrite(LED,LOW);
  }

  if (!client.connected()) {
    reconnect();
  }
  client.loop();

  unsigned long now = millis();
  if (now - lastMsg > 2000) { //perintah publish data
    lastMsg = now;
    TempAndHumidity  data = dht.getTempAndHumidity();

    String temp = String(data.temperature, 2);
    client.publish("/Thinkitive/temp", temp.c_str()); // publish temp topic /ThinkIOT/temp
    String hum = String(data.humidity, 1);
    client.publish("/Thinkitive/hum", hum.c_str());   // publish hum topic /ThinkIOT/hum

    Serial.print("Temperature: ");
    Serial.println(temp);
    Serial.print("Humidity: ");
    Serial.println(hum);
  }

  delay(1000);
}
```

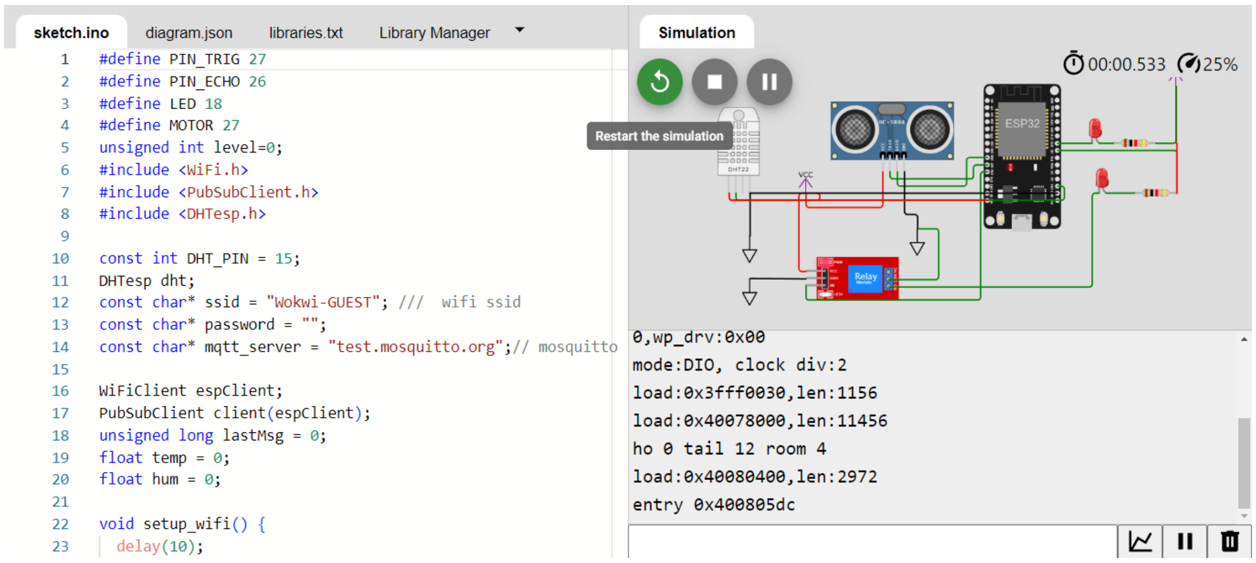
5.2 BACK END SOURCE CODE

```
{
  "version": 1,
  "author": "Kausik T",
  "editor": "wokwi",
"parts": [
  { "type": "wokwi-esp32-devkit-v1", "id": "esp", "top": -43.3, "left": 71.8, "attrs": {} },
  {
    "type": "wokwi-hc-sr04",
    "id": "ultrasonic1",
    "top": -17.7,
    "left": -138.5,
    "attrs": { "distance": "322" }
  },
  { "type": "wokwi-gnd", "id": "gnd1", "top": 153.6, "left": -29.4, "attrs": {} },
  { "type": "wokwi-vcc", "id": "vcc1", "top": 77.56, "left": -182.4, "attrs": {} },
  { "type": "wokwi-relay-module", "id": "relay1", "top": 192.2, "left": -172.8, "attrs": {} },
  { "type": "wokwi-gnd", "id": "gnd2", "top": 220.8, "left": -259.8, "attrs": {} },
  { "type": "wokwi-dht22", "id": "dht1", "top": -9.3, "left": -293.4, "attrs": {} },
  { "type": "wokwi-gnd", "id": "gnd3", "top": 163.2, "left": -259.8, "attrs": {} },
  { "type": "wokwi-led", "id": "led1", "top": 63.6, "left": 215, "attrs": { "color": "red" } },
  {
    "type": "wokwi-resistor",
    "id": "r1",
    "top": 99.95,
    "left": 278.4,
    "attrs": { "value": "1000" }
  },
  { "type": "wokwi-vcc", "id": "vcc2", "top": -66.44, "left": 326.4, "attrs": {} },
  { "type": "wokwi-led", "id": "led2", "top": -3.6, "left": 205.4, "attrs": { "color": "red" } },
  {
    "type": "wokwi-resistor",
    "id": "r2",
    "top": 32.75,
    "left": 249.6,
    "attrs": { "value": "1000" }
  }
],
"connections": [
  [ "esp:TX0", "$serialMonitor:RX", "", [] ],
  [ "esp:RX0", "$serialMonitor:TX", "", [] ],
    [ "ultrasonic1:TRIG", "esp:D27", "green", [ "v9.6", "h114.8", "v-9.6" ] ],
    [ "ultrasonic1:ECHO", "esp:D26", "green", [ "v19.2", "h95.2", "v-38.4" ] ],
    [ "ultrasonic1:GND", "gnd1:GND", "black", [ "v57.6", "h18" ] ],
    [ "vcc1:VCC", "ultrasonic1:VCC", "red", [ "v19.2", "h-28.8" ] ],
    [ "relay1:VCC", "vcc1:VCC", "red", [ "h-9.6", "v-105.6" ] ],
    [ "gnd2:GND", "relay1:GND", "black", [ "v0" ] ],
    [ "relay1:IN", "esp:D14", "green", [ "h0", "v19", "h240", "v-172.8" ] ],
    [ "dht1:GND", "gnd3:GND", "black", [ "v0" ] ],
    [ "dht1:VCC", "vcc1:VCC", "red", [ "v9.6", "h124.8", "v-9.6" ] ],
    [ "relay1:COM", "gnd1:GND", "green", [ "h68.4", "v-68.6", "h-28.8" ] ],
    [ "led1:A", "r1:1", "green", [ "v0" ] ],
    [ "r1:2", "esp:D5", "green", [ "v0" ] ],
    [ "relay1:NO", "led1:C", "green", [ "h279.6", "v-126.6" ] ],
    [ "vcc2:VCC", "r1:2", "red", [ "v76.8", "h1.2" ] ],
    [ "dht1:GND", "esp:GND.1", "black", [ "v0" ] ],
    [ "dht1:SDA", "esp:D15", "green", [ "v9.6", "h451.3", "v-19.2" ] ],
    [ "dht1:VCC", "esp:VIN", "red", [ "v0" ] ],
    [ "led2:A", "r2:1", "green", [ "v0", "h19.2" ] ],
    [ "esp:D18", "led2:C", "green", [ "h47.7" ] ],
    [ "r2:2", "vcc2:VCC", "green", [ "v0", "h27.6", "v0", "h0", "v0", "h0", "v0", "h0", "v0" ] ]
  ],
  "dependencies": {}
}
```

CHAPTER 6

SAMPLE OUTPUT

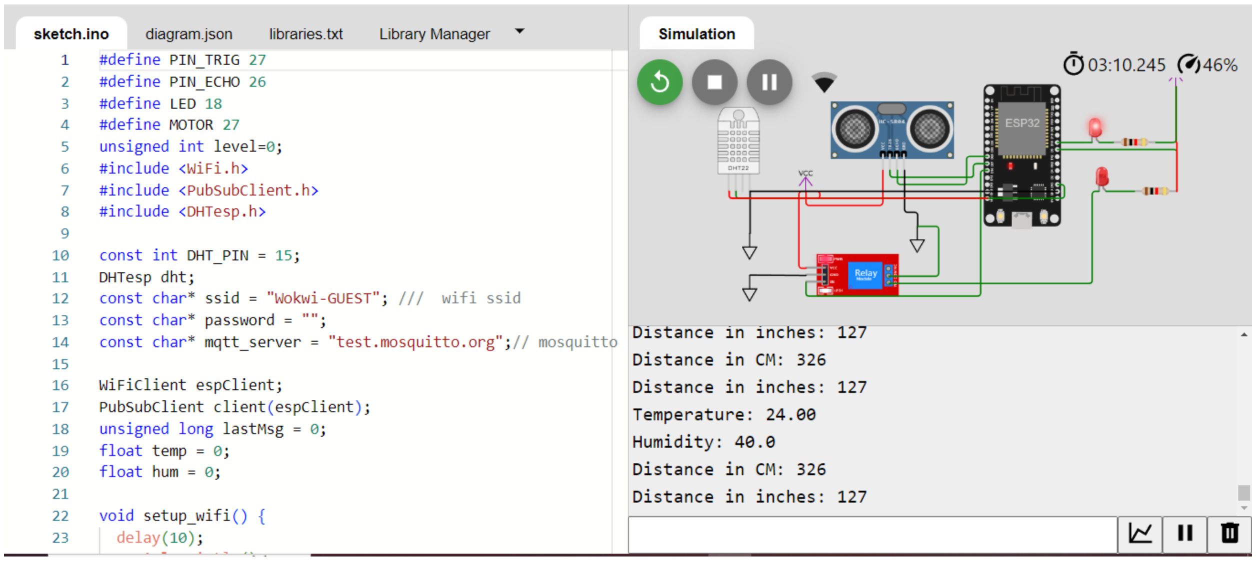
6.1 SCREENSHOT:



```
#define PIN_TRIG 27
#define PIN_ECHO 26
#define LED 18
#define MOTOR 27
unsigned int level=0;
#include <WiFi.h>
#include <PubSubClient.h>
#include <DHTesp.h>

const int DHT_PIN = 15;
DHTesp dht;
const char* ssid = "Wokwi-GUEST"; ///  wifi ssid
const char* password = "";
const char* mqtt_server = "test.mosquitto.org";// mosquitto

WiFiClient espClient;
PubSubClient client(espClient);
unsigned long lastMsg = 0;
float temp = 0;
float hum = 0;

void setup_wifi() {
  delay(10);
```

```
0,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0030,len:1156
load:0x40078000,len:11456
ho 0 tail 12 room 4
load:0x40080400,len:2972
entry 0x400805dc
```

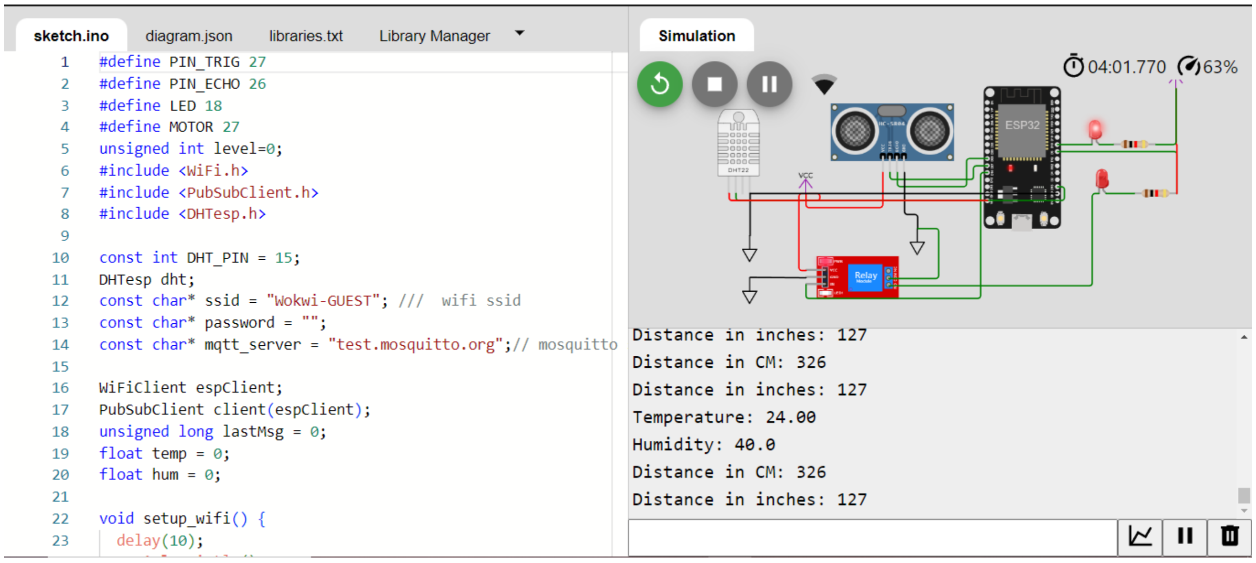
Fig:6.3.1



```
#define PIN_TRIG 27
#define PIN_ECHO 26
#define LED 18
#define MOTOR 27
unsigned int level=0;
#include <WiFi.h>
#include <PubSubClient.h>
#include <DHTesp.h>

const int DHT_PIN = 15;
DHTesp dht;
const char* ssid = "Wokwi-GUEST"; ///  wifi ssid
const char* password = "";
const char* mqtt_server = "test.mosquitto.org";// mosquitto

WiFiClient espClient;
PubSubClient client(espClient);
unsigned long lastMsg = 0;
float temp = 0;
float hum = 0;

void setup_wifi() {
  delay(10);
```

```
Distance in inches: 127
Distance in CM: 326
Distance in inches: 127
Temperature: 24.00
Humidity: 40.0
Distance in CM: 326
Distance in inches: 127
```

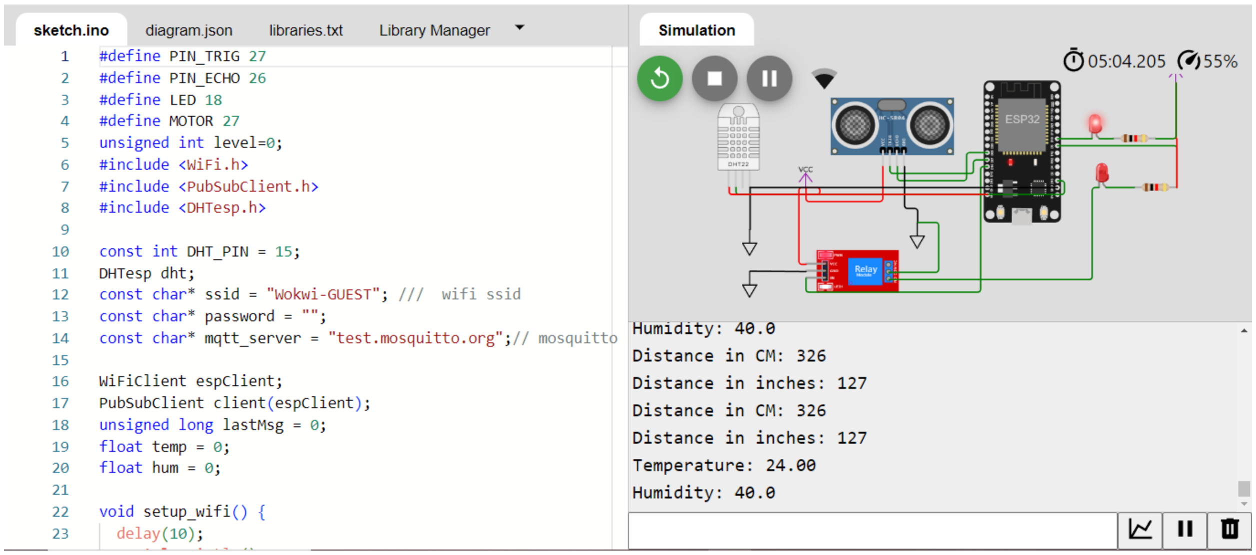
Fig:6.3.2



```
#define PIN_TRIG 27
#define PIN_ECHO 26
#define LED 18
#define MOTOR 27
unsigned int level=0;
#include <WiFi.h>
#include <PubSubClient.h>
#include <DHTesp.h>

const int DHT_PIN = 15;
DHTesp dht;
const char* ssid = "Wokwi-GUEST"; ///  wifi ssid
const char* password = "";
const char* mqtt_server = "test.mosquitto.org";// mosquitto

WiFiClient espClient;
PubSubClient client(espClient);
unsigned long lastMsg = 0;
float temp = 0;
float hum = 0;

void setup_wifi() {
  delay(10);
```

```
Distance in inches: 127
Distance in CM: 326
Distance in inches: 127
Temperature: 24.00
Humidity: 40.0
Distance in CM: 326
Distance in inches: 127
```

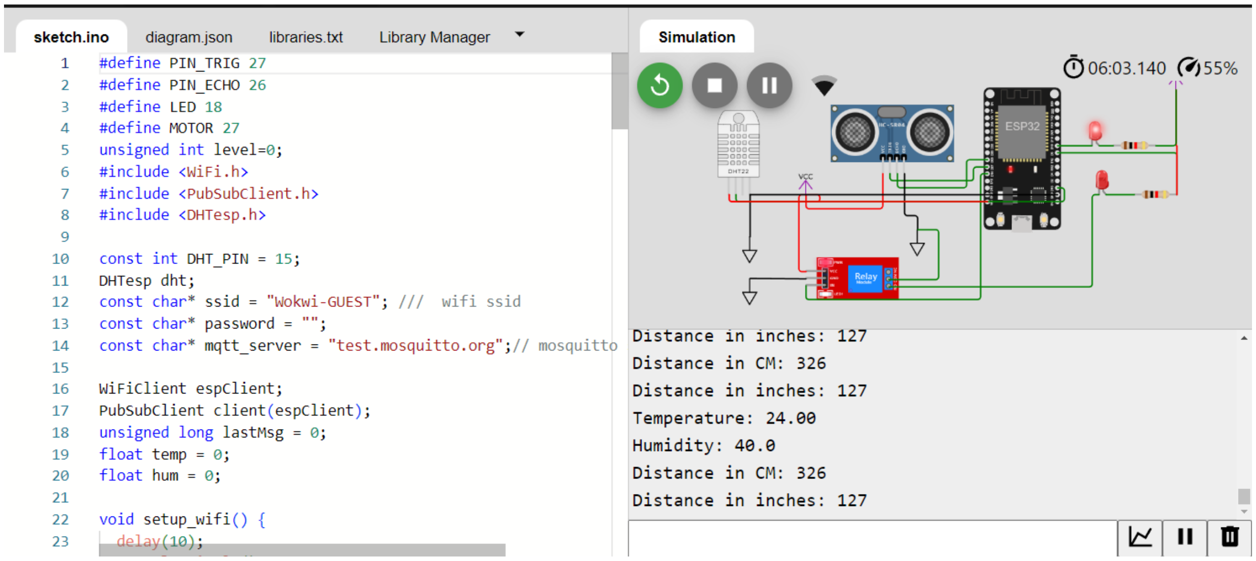
Fig:6.3.3



```
#define PIN_TRIG 27
#define PIN_ECHO 26
#define LED 18
#define MOTOR 27
unsigned int level=0;
#include <WiFi.h>
#include <PubSubClient.h>
#include <DHTesp.h>

const int DHT_PIN = 15;
DHTesp dht;
const char* ssid = "Wokwi-GUEST"; ///  wifi ssid
const char* password = "";
const char* mqtt_server = "test.mosquitto.org";// mosquitto

WiFiClient espClient;
PubSubClient client(espClient);
unsigned long lastMsg = 0;
float temp = 0;
float hum = 0;

void setup_wifi() {
  delay(10);
```

```
Humidity: 40.0
Distance in CM: 326
Distance in inches: 127
Distance in CM: 326
Distance in inches: 127
Temperature: 24.00
Humidity: 40.0
```

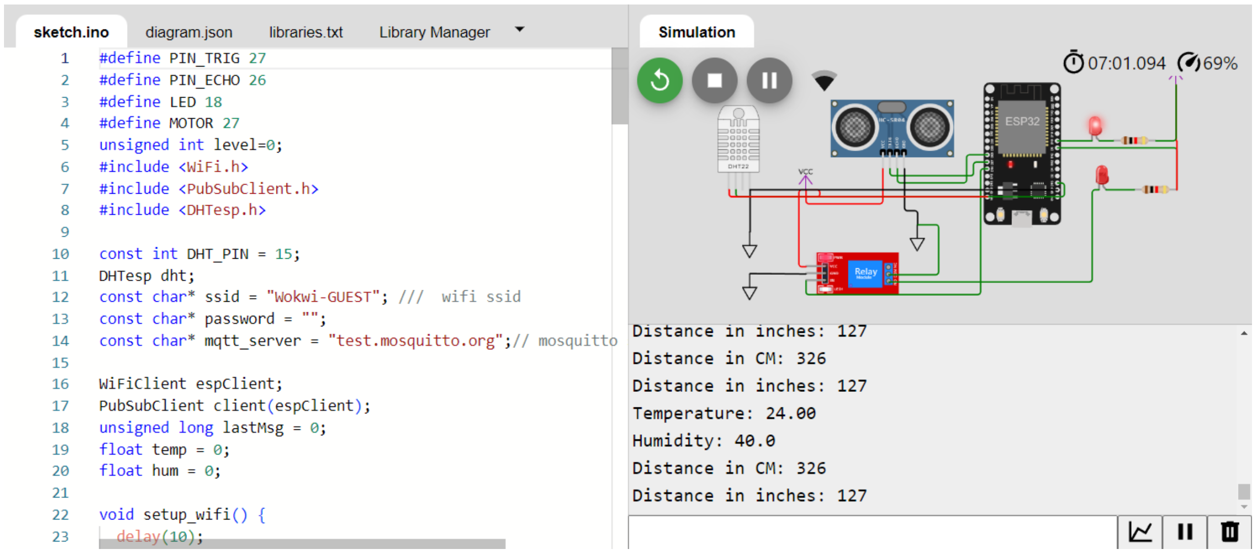
Fig:6.3.4



```
#define PIN_TRIG 27
#define PIN_ECHO 26
#define LED 18
#define MOTOR 27
unsigned int level=0;
#include <WiFi.h>
#include <PubSubClient.h>
#include <DHTesp.h>

const int DHT_PIN = 15;
DHTesp dht;
const char* ssid = "Wokwi-GUEST"; ///  wifi ssid
const char* password = "";
const char* mqtt_server = "test.mosquitto.org";// mosquitto

WiFiClient espClient;
PubSubClient client(espClient);
unsigned long lastMsg = 0;
float temp = 0;
float hum = 0;

void setup_wifi() {
  delay(10);
```

```
Distance in inches: 127
Distance in CM: 326
Distance in inches: 127
Temperature: 24.00
Humidity: 40.0
Distance in CM: 326
Distance in inches: 127
```

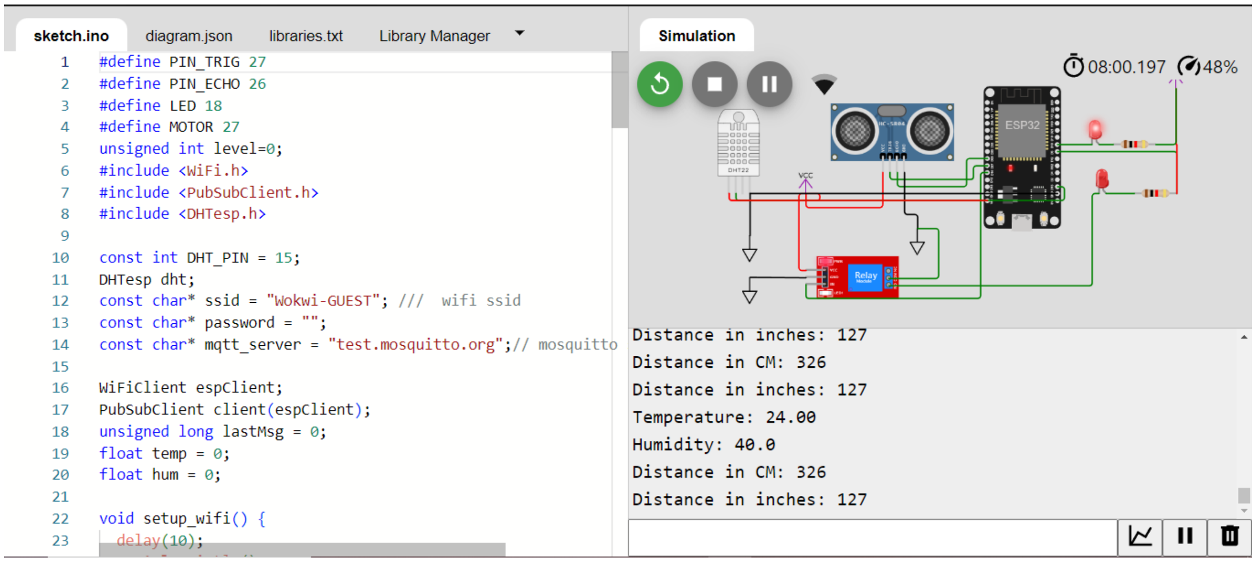
Fig:6.3.5



```
#define PIN_TRIG 27
#define PIN_ECHO 26
#define LED 18
#define MOTOR 27
unsigned int level=0;
#include <WiFi.h>
#include <PubSubClient.h>
#include <DHTesp.h>

const int DHT_PIN = 15;
DHTesp dht;
const char* ssid = "Wokwi-GUEST"; ///  wifi ssid
const char* password = "";
const char* mqtt_server = "test.mosquitto.org";// mosquitto

WiFiClient espClient;
PubSubClient client(espClient);
unsigned long lastMsg = 0;
float temp = 0;
float hum = 0;

void setup_wifi() {
  delay(10);
```

```
Distance in inches: 127
Distance in CM: 326
Distance in inches: 127
Temperature: 24.00
Humidity: 40.0
Distance in CM: 326
Distance in inches: 127
```

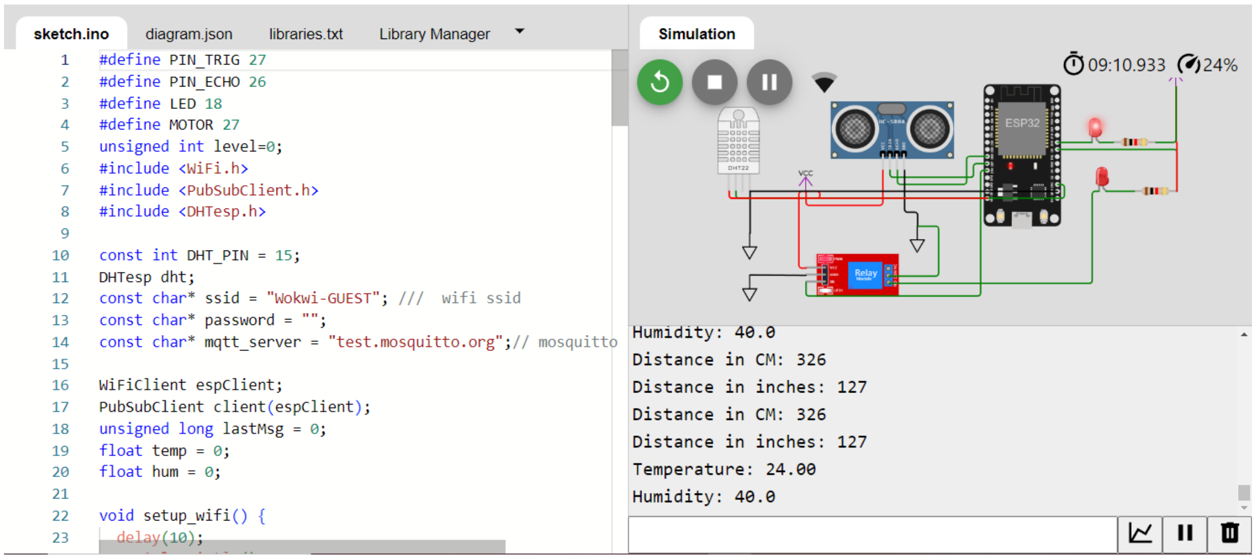
Fig:6.3.6



```
#define PIN_TRIG 27
#define PIN_ECHO 26
#define LED 18
#define MOTOR 27
unsigned int level=0;
#include <WiFi.h>
#include <PubSubClient.h>
#include <DHTesp.h>

const int DHT_PIN = 15;
DHTesp dht;
const char* ssid = "Wokwi-GUEST"; ///  wifi ssid
const char* password = "";
const char* mqtt_server = "test.mosquitto.org";// mosquitto

WiFiClient espClient;
PubSubClient client(espClient);
unsigned long lastMsg = 0;
float temp = 0;
float hum = 0;

void setup_wifi() {
  delay(10);
```

```
Distance in inches: 127
Distance in CM: 326
Distance in inches: 127
Temperature: 24.00
Humidity: 40.0
Distance in CM: 326
Distance in inches: 127
```

Fig6.3.7



```
#define PIN_TRIG 27
#define PIN_ECHO 26
#define LED 18
#define MOTOR 27
unsigned int level=0;
#include <WiFi.h>
#include <PubSubClient.h>
#include <DHTesp.h>

const int DHT_PIN = 15;
DHTesp dht;
const char* ssid = "Wokwi-GUEST"; ///  wifi ssid
const char* password = "";
const char* mqtt_server = "test.mosquitto.org";// mosquitto

WiFiClient espClient;
PubSubClient client(espClient);
unsigned long lastMsg = 0;
float temp = 0;
float hum = 0;

void setup_wifi() {
  delay(10);
```

```
Humidity: 40.0
Distance in CM: 326
Distance in inches: 127
Distance in CM: 326
Distance in inches: 127
Temperature: 24.00
Humidity: 40.0
```

Fig:6.3.8

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENT

Smart water fountains have the power to revolutionize hydration by providing a safer, more convenient, and personalized drinking experience. With their sustainable practices and inclusive designs, these fountains promote a healthier lifestyle and contribute to a greener environment. Let's embrace the potential of smart water fountains and ensure access to clean drinking water for all!

The future of smart water fountains holds endless possibilities. With advancements in artificial intelligence and Internet of Things, we can expect even more intelligent features like automatic bottle refills, personalized hydration reminders, and real-time water consumption data. Get ready for a future where staying hydrated is easier than ever!

CHAPTER 8

REFERENCE

1. Xavier Gibert, Vishal M Patel, and Rama Chellappa, in their IEEE paper titled as “Deep Multi-Task Learning for Railway Track Inspection” Volume 18, Issue 1, Jan 2017, pp 153 – 167.

2. S Mohamed Ashiq, K Karthikeyan, S Karthikeyan. “Fabrication of SemiAutomated Pressurized Flushing System in Indian Railway Toilet”, International Journal of Engineering and Advanced Technology (IJEAT), Volume-2, Issue-3, February2013.

3. Dr. ManojHedaoo, Dr. SuchitaHirde ,Ms. Arshi Khan “Sanitation In Indian Railway Premises: A Great Cause Of Concern”, International Journal of Advanced Engineering Technology, Mar 2012, Volume 3, Issue 1, pp 50 -55.

4. Dhanajay G Dange, Dattaprakash G Vernekar, Sagar D Kurhade, Prashant D Agwane, “Methodology for Design and Fabrication of Human Waste Disposal System for Indian Railway”, International Journal of Science Technology & Engineering, Volume 2, Issue 07, January2016, pp 14 – 19.

5. Mesch, F., Puente Le´on, F. &Engelberg, T., Train-based location by detecting rail switches. Computers in Railways VII, eds. J. Allen, R.J. Hill, C.A. Brebbia, G. Sciutto & S. Sone, WIT Press, Southampton, pp. 1251– 1260

6. K. Osathanunkul, K. Hantarkul, P. Pramokchon, P. Khoenkaw and N. Tantitharanukul, “Design and Implementation of an Automatic Smart Urinal Flusher”,International Computer Science and Engineering Conference (ICSEC2016), Chiang Mai, Thailand, Dec,2016, pp 14-17.

7. J. Shah and B. Mishra, “IoT enabled Environmental Monitoring System for Smart Cities”, International Conference on Internet of Things and Applications (IOTA), Maharashtra Institue of Technology, Pune, India, Volume 3, Issue 2, Jan 2016, pp383- 388.

8. K. Hantrakul, P. Pramokchon, P. Khoenkaw, N. Tantitharanukul, and K. Osathanunkul, “Automatic Faucet with Changeable Flow based on MQTT protocol”,International Computer Science and Engineering Conference (ICSEC2016), Chiang Mai, Thailand, 14-17 Dec,2016.

9. “Smart toilets using turbidity sensor” in international journal of innovative technology and exploring engineering (IJITEE) ISSN:2278-3075, volume-8 Issue5S March,2019, ES345501839/19©BEIESP. By Mithya V, Divya Prabha.N, Sisma Samlein S, Madhumitha M .

10. “Smart toilet using BLE beacon technology” in proceeding of the International Conference and Electronics systems(ICCES 2018) IEEE Xplore part number CFP18AO-ART; ISBN:978-1-5386—4765-3, By Ms. Nidhi R Mishra, Mr. Paras M Suri, Dr.(Mrs.) Shalu Chopra.

11. 2015 International conference on control, Instrumentation, communication and Computational Technologies (ICCICCT)-“GPS enabled Employee Registration and Attendance Tracing System”, IEEE paper.

12. International Journal of computer applications (0975- 8887)- “vehicle Tracking, Monitoring and Alerting System: A Review” Volume 119 – No.10, June 2015

13. Sneha Jangid, Sandeep Sharam, “An embedded system model for air quality monitoring,” 2016 International conference on computing for sustainable global development (India.Com), school of ICT, Gautam Buddha University Greater Noida, India.

14. Xavier Gibert, Vishal M Patel, Rama Chellappa, in their IEEE paper titled as “Deep Muiti-task Learning for Railway Track Inspection” Volume 18, Issue1, jan 2017, pp 153-167

15. . A.D Kadge, A. K. Varute, P. G. Patil, P. R. Belukhi “Automatic sewage disposal system for train”, International journal of Emerging Research in management and technology (Volume- 5, Issue-5), May