SMART WATER FOUNTAINS

Introduction:

The Smart Water Fountain is an IoT (Internet of Things) device designed to provide convenient and sustainable access to clean drinking water while promoting water conservation. This project combines technology, design thinking, and environmental consciousness to tackle water-related challenges faced by communities globally. It offers an intelligent, data-driven solution to not only deliver clean water but also track and manage water usage efficiently.

Problem Definition:

Access to clean and safe drinking water is a fundamental human right, yet many people around the world still lack easy access to potable water sources. Even in developed areas, there is often a lack of awareness regarding water consumption, leading to wastage. To address these issues, the Smart Water Fountain project aims to create an innovative solution that promotes responsible water

consumption and ensures access to clean drinking water in public spaces.

Objectives:

Clean Water Access: Ensure easy access to safe and clean drinking water in public spaces, promoting hydration and reducing the need for single-use plastic bottles. Water Conservation: Encourage responsible water consumption through real-time monitoring, feedback mechanisms, and gamification elements. Data-driven Insights: Collect and analyse data on water consumption patterns to identify opportunities for further conservation and optimization. User-friendly Design: Create an intuitive, aesthetically pleasing water fountain design that attracts users and is accessible to people of all abilities.

IoT device setup:

Technology Selection

- **Research IoT Platforms**: Look for user-friendly and cost-effective IoT platforms like Arduino, Raspberry Pi, or ESP8266/ESP32 to build the fountain's hardware.

Hardware Assembly

- **Assemble the Hardware:** Start with a basic water fountain design. Attach a water pump to a water source and connect it to your chosen microcontroller (Arduino or Raspberry Pi).

Sensor Integration

- **Add Water Flow Sensor:** Integrate a water flow sensor into the water line to measure water usage.

- **Temperature Sensor:** Include a temperature sensor to monitor water quality.

Software Development

- **Write Code:** Develop simple code (in Arduino IDE or Python) to control the water pump and collect data from the sensors.

- **User Interface:** Create a basic user interface using a basic web page for user interaction.

Platform used: Wokwi simulator.

CODE IMPLEMENTATION:

main.py

```
import machine
import time

# Pin assignments for the ultrasonic sensor
TRIGGER_PIN = 23  # GPIO23 for trigger
ECHO_PIN = 22     # GPIO22 for echo

# Pin assignment for the LED
LEAK_LED_PIN = 19  # GPIO19 for the LED

# Set the pin modes
trigger = machine.Pin(TRIGGER_PIN, machine.Pin.OUT)
echo = machine.Pin(ECHO_PIN, machine.Pin.IN)
leak_led = machine.Pin(LEAK_LED_PIN, machine.Pin.OUT)

# Function to measure distance using the ultrasonic sensor
def measure_distance():
    # Generate a short trigger pulse
    trigger.value(0)
    time.sleep_us(5)
    trigger.value(1)
    time.sleep_us(10)
    trigger.value(0)

    # Measure the echo pulse duration to calculate distance
    pulse_start = pulse_end = 0
    while echo.value() == 0:
        pulse_start = time.ticks_us()
    while echo.value() == 1:
        pulse_end = time.ticks_us()
```

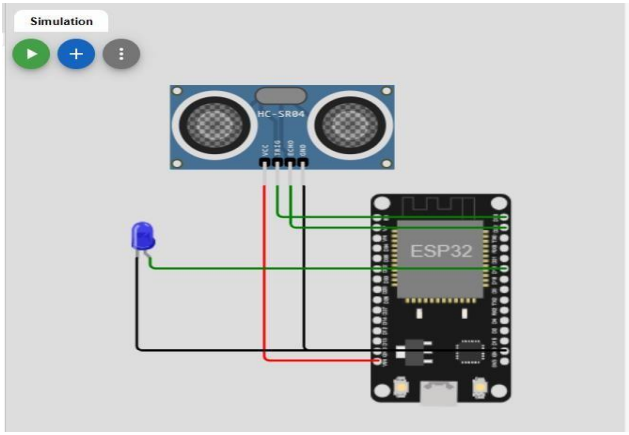
```
    pulse_duration = pulse_end - pulse_start

    # Calculate distance in centimeters (assuming the speed of sound
    distance = (pulse_duration * 0.0343) / 2  # Divide by 2 for one-

    return distance

# Function to check for a water leak
def check_for_leak():
    # Measure the distance from the ultrasonic sensor
    distance = measure_distance()

    # Set the threshold distance for detecting a leak (adjust as nee
    threshold_distance = 10  # Adjust this value based on your tank

    if distance < threshold_distance:
        # If the distance is less than the threshold, a leak is dete
        return True
    else:
        return False

# Main loop
while True:
    if check_for_leak():
        # Blink the LED to indicate a leak
        leak_led.value(1)  # LED ON
        time.sleep(0.5)
        leak_led.value(0)  # LED OFF
        time.sleep(0.5)

    else:
        leak_led.value(0)  # LED OFF

    time.sleep(1)  # Delay between measurements
```

diagram.json

diagram.json | Library Manager

```
{
  "version": 1,
  "author": "Uri Shaked",
  "editor": "wokwi",
  "parts": [
    { "type": "wokwi-esp32-devkit-v1", "id": "esp", "top": -14.5, "left": 81.4, "attrs": {} },
    { "type": "wokwi-led", "id": "led2", "top": 6, "left": -111.4, "attrs": { "color": "blue" } },
    { "type": "wokwi-hc-sr04", "id": "ultrasonic1", "top": -113.7, "left": -71.3, "attrs": {} }
  ],
  "connections": [
    [ "esp:TX0", "$serialMonitor:RX", "", [] ],
    [ "esp:RX0", "$serialMonitor:TX", "", [] ],
    [ "ultrasonic1:GND", "esp:GND.2", "black", [ "v0" ] ],
    [ "ultrasonic1:ECHO", "esp:D22", "green", [ "v0" ] ],
    [ "ultrasonic1:TRIG", "esp:D23", "green", [ "v0" ] ],
    [ "ultrasonic1:VCC", "esp:VIN", "red", [ "v0" ] ],
    [ "led2:A", "esp:D19", "green", [ "v0" ] ],
    [ "led2:C", "esp:GND.1", "black", [ "v0" ] ]
  ],
  "dependencies": {}
}
```

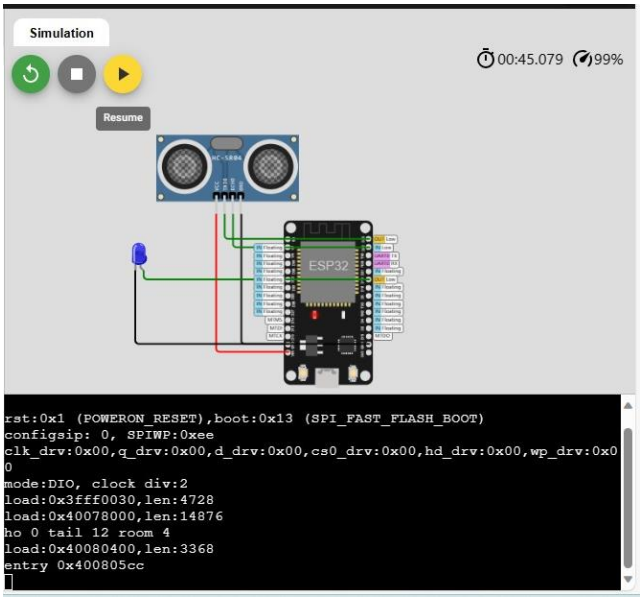
CIRCUIT DIAGRAM:



STIMULATION OUTPUT:

main.py | diagram.json | Library Manager

```
import machine
import time

# Pin assignments for the ultrasonic sensor
TRIGGER_PIN = 23  # GPIO23 for trigger
ECHO_PIN = 22     # GPIO22 for echo

# Pin assignment for the LED
LEAK_LED_PIN = 19  # GPIO19 for the LED

# Set the pin modes
trigger = machine.Pin(TRIGGER_PIN, machine.Pin.OUT)
echo = machine.Pin(ECHO_PIN, machine.Pin.IN)
leak_led = machine.Pin(LEAK_LED_PIN, machine.Pin.OUT)

# Function to measure distance using the ultrasonic sensor
def measure_distance():
    # Generate a short trigger pulse
    trigger.value(0)
    time.sleep_us(5)
    trigger.value(1)
    time.sleep_us(10)
    trigger.value(0)

    # Measure the echo pulse duration to calculate distance
    pulse_start = pulse_end = 0
    while echo.value() == 0:
        pulse_start = time.ticks_us()
    while echo.value() == 1:
        pulse_end = time.ticks_us()
```



PLATFORM DEVELOPMENT:

Webpage code:

HTML:

```
<!DOCTYPE html>
<html>
<head>
    <title>Water Fountain Status
Platform</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <h1>Water Fountain Status</h1>

    <div id="water-fountain-status">
        <table>
            <thead>
                <tr>
                    <th>Fountain Name</th>
                    <th>Flow Rate</th>
                    <th>Status</th>
                </tr>
            </thead>
            <tbody>
                <tr>
                    <td></td>
```

```
                    <td></td>
                    <td></td>
                </tr>
                <tr>
                    <td></td>
                    <td></td>
                    <td></td>
                </tr>
                <tr>
                    <td></td>
                    <td></td>
                    <td></td>
                </tr>
                <tr>
                    <td></td>
                    <td></td>
                    <td></td>
                </tr>
                <tr>
                    <td></td>
                    <td></td>
                    <td></td>
                </tr>
            </tbody>
        </table>
    </div>
<script src="script.js"></script>
</body>
</html>
```

Styles.css

```
body {
    font-family: sans-serif;
}

h1 {
    text-align: center;
}

#water-fountain-status {
    width: 500px;
    margin: 0 auto;
    border: 1px solid black;
    padding: 10px;
}

#water-fountain-status table {
    width: 100%;
    border-collapse: collapse;
}

#water-fountain-status th, td {
    border: 1px solid black;
    padding: 5px;
}

td, th {
    color: #ff0000;
  }

h1 {
    font-weight: bold;
```

```
    text-align: center;
  }

body {
    background-color: #f5f5f5;
  }

td {
    height: 10px;
  }
```

Script.js

```
// This function would be used to update the
water fountain status table with the latest
data
function updateWaterFountainStatus() {
    // Get the water fountain status data from
the server
    const waterFountainStatusData =
getWaterFountainStatusData();

    // Get the water fountain status table
    const tableBody =
document.querySelector("#water-fountain-status
tbody");
```

```
    // Iterate over the water fountain status
data
    for (let i = 0; i <
waterFountainStatusData.length; i++) {
      const waterFountainStatus =
waterFountainStatusData[i];

      // Get the existing table row for the
water fountain, if it exists
      const existingTableRow =
tableBody.querySelector(`tr[data-fountain-
name="${waterFountainStatus.fountainName}"]`);

      // If the existing table row does not
exist, then create a new row
      if (!existingTableRow) {
        const newTableRow =
document.createElement("tr");
        newTableRow.setAttribute("data-
fountain-name",
waterFountainStatus.fountainName);
        tableBody.appendChild(newTableRow);
      }

      // Update the existing table row with the
latest data
      const fountainNameCell =
existingTableRow.querySelector("td:first-
child");
      fountainNameCell.textContent =
waterFountainStatus.fountainName;
```

```
      const flowRateCell =
existingTableRow.querySelector("td:nth-
child(2)");
      flowRateCell.textContent =
waterFountainStatus.flowRate;

      const statusCell =
existingTableRow.querySelector("td:nth-
child(3)");
      statusCell.textContent =
waterFountainStatus.status;

      const countCell =
existingTableRow.querySelector("td:last-
child");
      countCell.textContent =
waterFountainStatus.count;
    }
  }

const tableBody =
document.querySelector("#water-fountain-status
tbody");

  // Add 7 empty rows to the table
  for (let i = 0; i < 7; i++) {
    const emptyTableRow =
document.createElement("tr");
    tableBody.appendChild(emptyTableRow);
  }
```

```
function getWaterFountainStatusData(){
    // TODO: Implement this function to get the
water fountain status data from the server.
    // For example, you could use an AJAX
request to fetch the data from a server.
    var waterFountainStatusData = [
      {
        "fountainName": "Fountain 1",
        "flowRate": 100,
        "status": "OK",
        "count": 2
      },
      {
        "fountainName": "Fountain 2",
        "flowRate": 50,
        "status": "Alert",
        "count": 1
      }
    ];
    return waterFountainStatusData;
  }


// Update the water fountain status table every
5 seconds
setInterval(updateWaterFountainStatus, 5000);
```

OUTPUT WEBSITE: