NAAN MUDHALVAN

PROJECT PHASE 1

Problem Definition:

Project Title: Smart Water Fountain

Problem Statement:

The current water fountains in our public spaces lack user engagement and efficient water usage. There is a need for a smart water fountain solution that not only provides refreshing drinking water but also incorporates technology to enhance user experience, reduce water wastage, and promote sustainability.

It seems like there might be a slight mix-up in your project description and design thinking approach. The project aims to enhance public water fountains by implementing IoT sensors to control water flow and detect malfunctions. The primary objective is to provide real-time information about water fountain status to residents through a public platform. This project includes defining objectives, designing the IoT sensor system, developing the water fountain status platform, and integrating them using IoT technology and Python.

❖ Project Objectives:

- ❖ Enhanced User Experience: Create a water fountain design that engages users and provides an enjoyable experience.
- ❖ Water Conservation: Develop a system that efficiently dispenses water to minimize waste.
- ❖ Sustainability: Implement eco-friendly features such as water purification and usage monitoring.
- ❖ Accessibility: Ensure that the smart water fountain is accessible to people of all abilities.
- ❖ maintenance and Monitoring: Establish a system for remote monitoring and maintenance of the fountains.
- ❖ Project Objectives: Define objectives such as real-time water fountain monitoring, efficient water usage, malfunction detection, and resident awareness.
- ❖ IoT Sensor Design: Plan the deployment of IoT sensors (e.g., flow rate sensors, pressure sensors) in public water fountains.
- ❖ Real-Time Transit Information Platform: Design a mobile app interface that displays real-time parking availability to users.
- ❖ Integration Approach:Determine how IoT sensors will send data to the water fountain status platform

❖ Design Thinking Process:

- ❖ Empathize: Understand the needs and pain points of the users. Conduct surveys, interviews, and observations in public spaces to gather insights about how people interact with existing water fountains.

- Define: Define the problem statement and project objectives based on the insights gained during the empathize stage. Prioritize the most critical issues to address.
- Ideate: Brainstorm potential solutions to the defined problems. Encourage a diverse range of ideas from team members and stakeholders. Consider both technological and design innovations.
- Prototype: Create prototypes of the smart water fountain design. These can range from low-fidelity sketches to high-fidelity physical models or digital simulations. Test these prototypes with potential users to gather feedback.
- Test: Gather feedback from users on the prototypes. Identify what works and what needs improvement. Iterate on the design based on the feedback received.
- Implement: Develop a working prototype of the smart water fountain based on the refined design. Ensure that it incorporates the desired technological features such as sensors, purification systems, and user interfaces.
- Test Again: Test the fully functional prototype in real-world settings to evaluate its performance, user experience, and sustainability features.
- Launch: Once the smart water fountain meets all project objectives and requirements, launch it in selected public spaces. Monitor its performance and user feedback during this phase.
- Evaluate and Iterate: Continuously gather data on water usage, user satisfaction, and maintenance needs. Use this data to make iterative improvements to the smart water fountain design.

- Scale: If the initial deployment is successful, consider scaling the smart water fountain solution to more locations or collaborating with municipalities and organizations to implement it in broader contexts.

- Remember that the design thinking process is iterative, and you may need to revisit previous stages as you gather more information and insights. Additionally, involve stakeholders, including end-users, throughout the process to ensure the solution meets their needs and expectations.

Design Thinking:

The project to enhance public water fountains using IoT sensors and a real-time status platform sounds like a valuable initiative. Here's a step-by-step breakdown of how you might approach this project:

1. Define Objectives and Requirements:

- Clearly outline the goals and objectives of your project. What do you want to achieve with this enhanced water fountain system?

- Identify the key requirements, such as the number of water fountains to be upgraded, the types of sensors needed, and the desired features of the status platform.

2. IoT Sensor System Design:

- Select the appropriate IoT sensors for monitoring the water fountains. These could include flow sensors to control water flow and various sensors to detect malfunctions (e.g., water level, temperature, pressure, and leak detectors).

- Design the sensor network architecture, considering factors like sensor placement, power supply (battery or mains), and communication protocols (Wi-Fi, Bluetooth, LoRa, etc.).

- Plan how sensor data will be collected, processed, and transmitted to the platform.

3. Water Fountain Status Platform Development:

- Decide on the technology stack for developing the status platform. Python is a suitable choice for backend development due to its versatility.

- Develop a user-friendly web interface or mobile app for residents to access real-time information about the water fountains. Ensure that it's intuitive and easy to use.

- Implement a database to store historical data and track fountain status changes over time.

- Create features for administrators to monitor and manage the system, including setting alerts for malfunctions.

4. Integration Using IoT Technology:

- Write the firmware for the IoT sensors, allowing them to collect data and communicate with the central platform.
- Implement data transmission and security protocols to protect the integrity of sensor data during transmission.
- Set up a cloud-based or on-premises server to receive and process data from the sensors.
- Develop APIs or messaging systems for seamless communication between the sensors and the platform.

5. Testing and Validation:
   - Thoroughly test the entire system to ensure it works as intended. This includes testing sensor data accuracy, platform functionality, and the user interface.
   - Conduct field tests to assess the system's performance under real-world conditions, including variations in weather and usage patterns.

6. Deployment and Maintenance:
   - Install the IoT sensors on the selected water fountains and deploy the platform for residents to access.
   - Develop a maintenance plan to regularly check and update sensors, replace batteries if necessary, and troubleshoot any issues that may arise.
   - Continuously monitor the system's performance and gather user feedback to make improvements.

7. Data Analytics and Reporting:

- Consider implementing data analytics tools to gain insights from the collected sensor data. For example, you can identify trends, detect potential issues early, and optimize water usage.

- Generate regular reports for stakeholders and the public to showcase the benefits and impact of the enhanced water fountain system.

8. Documentation and Training:

- Document the entire project, including system architecture, sensor specifications, codebase, and maintenance procedures.

- Provide training for administrators and maintenance personnel to ensure they can effectively operate and troubleshoot the system.

9. Public Engagement:

- Promote the new water fountain system to the public, emphasizing its benefits, such as reduced water wastage, improved maintenance, and real-time information availability.

10. Scale and Expansion:

- Consider the potential for expanding the system to more water fountains or adding additional features based on user feedback and evolving needs.

Throughout the project, it's essential to prioritize data privacy and security, ensuring that sensitive information is protected. Collaboration with local authorities and relevant stakeholders can also be crucial for project success.

NAAN MUDHALVAN

PROJECT PHASE 2

Project Title: Smart Water Fountain

INNOVATION:

Consider incorporating predictive maintenance algorithms to identify potential malfunctions before they occur.

Creating a smart water fountain using IoT (Internet of Things) technology and incorporating predictive maintenance algorithms to ensure its reliability and functionality is a great idea. Below is a high-level algorithm to guide you through the process:

Step 1: Define Requirements and Objectives

1.1. Identify Use Cases:
Determine the primary use cases for your smart water fountain, such as providing clean drinking water, tracking usage, or enhancing user experience.

1.2. Define Objectives:
Clearly outline the goals you want to achieve with your smart water fountain, including predictive maintenance to prevent malfunctions.

Step 2: Hardware Selection and Setup

2.1. Select IoT Hardware:

Choose the necessary IoT hardware components, such as sensors (for water quality, flow rate, temperature, etc.), microcontroller (e.g., Raspberry Pi or Arduino), and connectivity modules (e.g., Wi-Fi, Bluetooth, or cellular).

2.2. Assemble Hardware:

Assemble the chosen hardware components and set up the physical structure of the water fountain.

Step 3: Data Collection

3.1. Sensor Data:

Implement sensors to collect relevant data, such as water quality, usage frequency, and environmental conditions (e.g., temperature and humidity).

3.2. Data Transmission:

Set up a data transmission mechanism to send sensor data to a central server or cloud platform for processing and analysis.

Step 4: Cloud Platform and Data Storage

4.1. Cloud Integration:

Choose a cloud platform (e.g., AWS, Azure, Google Cloud) for data storage and processing. Set up the necessary cloud infrastructure to handle incoming data.

4.2. Data Storage:

Store the collected sensor data securely in a database or data lake.

Step 5: Predictive Maintenance Algorithm

5.1. Data Preprocessing:

Clean and preprocess the collected data. This may involve filtering outliers and handling missing values.

5.2. Feature Engineering:

Extract relevant features from the data that can be used to predict potential malfunctions. This may include patterns in water quality, usage trends, and sensor data correlations.

5.3. Algorithm Selection:

Choose or develop a predictive maintenance algorithm. Common approaches include machine learning models (e.g., regression, classification, or time series forecasting) and anomaly detection algorithms (e.g., Isolation Forest, One-Class SVM).

5.4. Model Training:

Train the selected predictive maintenance algorithm using historical data, including instances of past malfunctions and maintenance records.

5.5. Real-time Prediction:

Implement the trained algorithm to make real-time predictions based on incoming sensor data. Set up alerts or notifications for potential malfunctions.

Step 6: User Interface

6.1. Dashboard:

Develop a user-friendly dashboard or mobile application that displays real-time information about the water fountain's status, water quality, and maintenance alerts.

6.2. User Interaction:

Enable users to interact with the fountain through the user interface, such as adjusting water flow or viewing usage history.

Step 7: Maintenance and Alerts

7.1. Alert Generation:

When the predictive maintenance algorithm identifies potential malfunctions or maintenance needs, generate alerts or notifications for maintenance staff or users.

7.2. Maintenance Workflow:

Implement a workflow for maintenance staff to respond to alerts, schedule maintenance, and document maintenance actions.

Step 8: Testing and Deployment

8.1. Testing:

Thoroughly test the smart water fountain system to ensure it functions as intended, including the predictive maintenance aspect.

8.2. Deployment:

Deploy the system in the desired location, ensuring connectivity, power supply, and data transmission are reliable.

Step 9: Monitoring and Continuous Improvement

9.1. Ongoing Monitoring:

Continuously monitor the system's performance and data quality. Update the predictive maintenance algorithm as needed to improve accuracy.

9.2. User Feedback:

Collect feedback from users and maintenance staff to identify areas for improvement and new features.

Step 10: Maintenance Records and Documentation

10.1. Maintain Records:

Keep detailed records of maintenance actions and system performance to aid in future improvements and troubleshooting.

By following these steps, you can create a smart water fountain that not only provides enhanced functionality but also incorporates predictive maintenance algorithms to ensure its long-term reliability and minimize downtime.

NAAN MUDHALVAN

PROJECT PHASE ***3***

Project Title: **Smart Water Fountains**

Development Part 1

In this part you will begin building your project. Start building the IoT-enabled Smart Water Fountains system. Deploy IoT sensors (e.g., flow rate sensors, pressure sensors) in public water fountains to monitor water flow and detect malfunctions.

Develop a Python script on the IoT sensors to send real-time water fountain status data to the platform.

- First step of developing a smart water fountain is to simulate it using wokwi simulator. Basic components needed for the smart water fountain simulation are listed below:

1)NODE MCU ESP32 – it act as microcontroller .

2)water pump – To pump water from the tank to fountain.

3)relay module – To control the water pump.

4)Ultrasonic sensor(HC SR04) – To detect the water level in the fountain.

5)wokwi virtual components – for web interface and Simulation.

- Then the circuit is build in Wokwi circuit editor by adding the components like NODE MCU ESP32,waterpump,relay module and ultrasonic sensor. A Button and range element is added from the virtual components which provide web interfaces control for the smart water fountain.

- In the circuit, relay module and ultrasonic sensors are connected to the NODE MCU ESP32(Microcontroller). The water pump is connected to relay module to control the water flow in the pump.

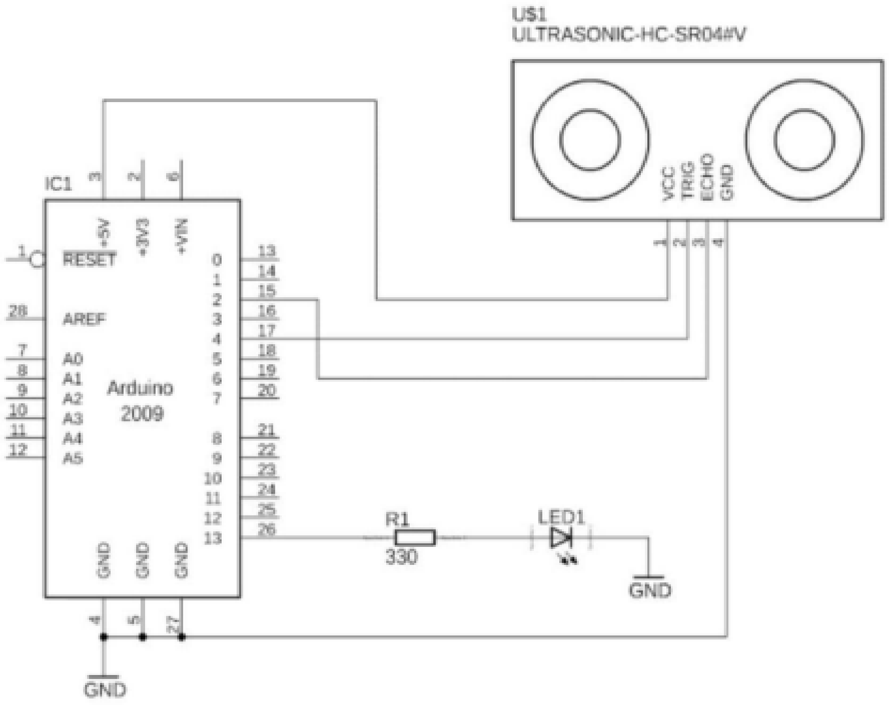
- Ultrasonic sensor was connected to the ESP32 to monitor the water level in fountain and update the user interface.

- Then write the python script for all the components using aurdiuno code editor.

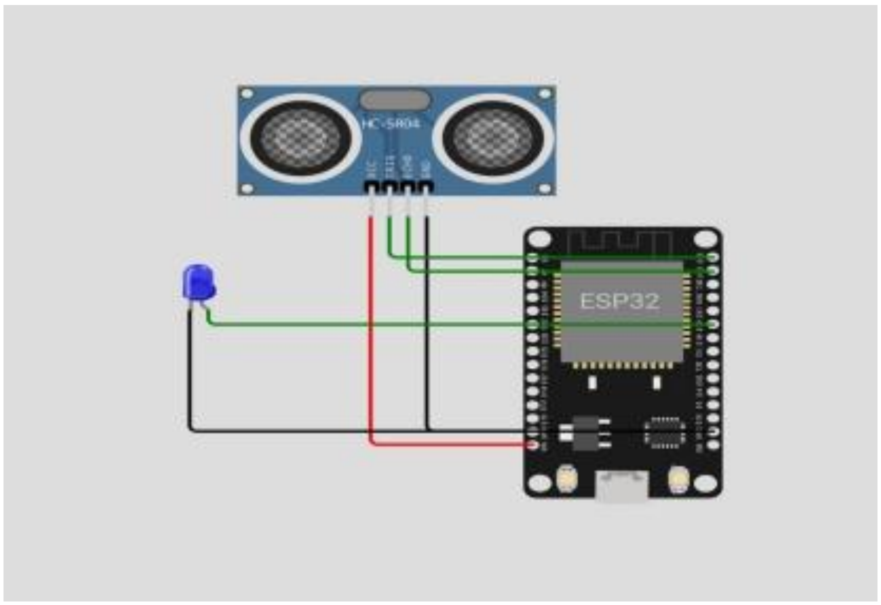
CODING:

```
#include <ESP32WiFi.h>
 #include <WiFiClient.h>
 #include < Ultrasonic.h>

 const char* ssid = "YourWiFiSSID";
 const char* password = "YourWiFiPassword";

const int trigPin = D2;
 const int echoPin = D3;
 const int relayPin = D1;

Ultrasonic ultrasonic(trigPin, echoPin);
WiFiServer server(80);

void setup() {
pinMode(relayPin, OUTPUT);
 digitalWrite(relayPin, LOW);
Serial.begin(115200);
WiFi.begin(ssid, password);
 while (WiFi.status() != WL_CONNECTED) {
 delay(1000);
 Serial.println("Connecting to WiFi...");
}
 server.begin();
}
 void loop()  {
WiFiClient client = server.available();
if (client) {
String request = client.readStringUntil('\r');
 if (request.indexOf("/on") != -1) {
 digitalWrite(relayPin, HIGH); // Turn the pump on
delay(2000); // Run the pump for 2 seconds
 digitalWrite(relayPin, LOW); // Turn the pump off
}
client.flush();
}
 float distance = ultrasonic.read();
 if (distance < 10) {
// Water is low, update the web interface
 // You can send an HTML response to the client here
}
}
```

- Then the circuit was saved and simulated. With this setup, we can simulate a smart water fountain that can be remotely controlled, and water level is monitored.



CircuitDiagramof water detection system



NAAN MUDHALVAN

PROJECT PHASE ***4***

Project Title: **Smart Water Fountains**

In this part you will continue building your project.

Continue building the project by developing the water fountain status platform.
Use web development technologies (e.g., HTML, CSS, JavaScript) to create a platform that displays real-time water fountain status.
Design the platform to receive and display real-time water fountain data, including water flow rate and malfunction alerts.

Creating a real-time water fountain status platform involves building a web application that can display and update data in real-time. To achieve this, you can use HTML, CSS, and JavaScript along with a backend server to provide the real-time data. Below is a simple example of how you can structure such a platform.

1. <u>HTML (index.html)</u>

```
<!DOCTYPE html>
<html>
<head>
    <title>Water Fountain Status</title>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
    <h1>Water Fountain Status</h1>
    <div id="status">
        <p>Flow Rate: <span id="flowRate">Updating...</span> liters per minute</p>
        <p>Malfunction Alert: <span id="malfunction">No</span></p>
    </div>
    <script src="script.js"></script>
</body>
</html>
```

2. CSS (styles.css)

```css
body {
   font-family: Arial, sans-serif;
   text-align: center;
   background-color: #f0f0f0;
}

h1 {
   color: #333;
}

#status {
   background-color: #fff;
   padding: 20px;
   border-radius: 10px;
   box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
   margin: 20px auto;
   width: 300px;
}

#flowRate {
   color: #009900;
   font-weight: bold;
}

#malfunction {
   color: #FF0000;
   font-weight: bold;
}
```

3. JavaScript (script.js)

```
function updateStatus() {
   // Simulate real-time data
   const flowRate = Math.random() * 10 + 1; // Random flow rate
between 1 and 10 liters per minute
   const isMalfunction = Math.random() < 0.1; // 10% chance of
malfunction

   document.getElementById("flowRate").textContent =
flowRate.toFixed(2);
   document.getElementById("malfunction").textContent =
isMalfunction ? "Yes" : "No";

   // Update every 5 seconds (you can adjust the interval)
   setTimeout(updateStatus, 5000);
}

// Initial update
updateStatus();
```

- This simple platform will display the water fountain's current status, including the water flow rate and whether there is a malfunction alert. It uses JavaScript to simulate real-time updates, but in a real application, you would replace the simulated data with actual data from your water fountain sensors or a backend server.

- To implement real-time data updates with actual sensor data, you would typically use technologies like WebSockets, AJAX, or Server-Sent Events (SSE) to receive and update data from the server. Additionally, you would need a backend server to collect and provide the data to the front-end. The choice of backend technology can depend on your specific requirements and constraints.

1. Flow Rate:

This is a simulated value representing a flow rate in Gallons Per Minute (GPM). It is updated every 2 seconds with a random value between 0 and 10, rounded to two decimal places (e.g., "4.25 GPM").

2. Malfunction Status:

This is a simulated status that indicates whether there's a malfunction or not. It's updated every 2 seconds with a 20% probability of showing "Yes" (indicating a malfunction) and an 80% probability of showing "No" (indicating no malfunction).

<u>OUTPUT:</u>(For javascript)

Flow Rate: 6.75 GPM
Malfunction Status: No

These values will change every 2 seconds as the `updateStatus` function is called repeatedly. You can place these values within HTML elements with the corresponding IDs (`'flow-rate-value'` and `'malfunction-status'`) to see the updates on your web page.

NAAN MUDHALVAN

PROJECT PHASE 5

Project Title: Smart Water Fountains

Phase 5: Project Documentation & Submission

In this part you will document your project and prepare it for submission.

Objectives:

The objective of the Smart Water Fountains project is to create an IoT-based system that monitors and displays the real-time status of water fountains in public spaces. This system aims to promote water efficiency by providing data-driven insights and increasing public awareness of water consumption.

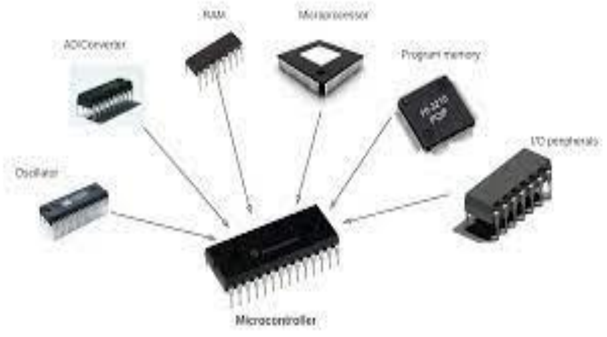
Creating a smart water fountain using IoT (Internet of Things) involves integrating sensors, microcontrollers, and connectivity to enable monitoring, control, and automation. Here’s an outline of how you could approach designing a smart water fountain using IoT:

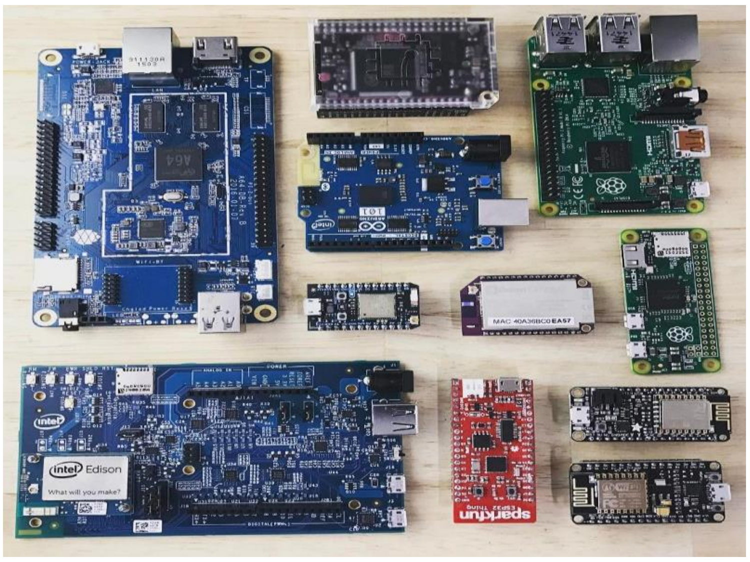
Components Needed:

1. Water Fountain: A basic or custom-designed fountain that you want to make 'smart'.

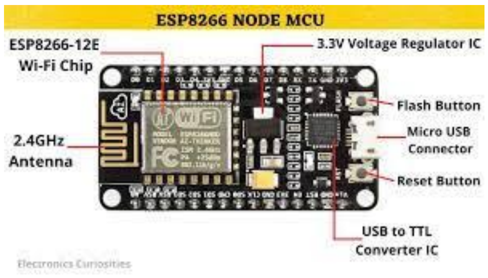
2. Water Level Sensor:Ultrasonic, capacitive, or other water level sensors to detect the water level in the fountain.

3. Microcontroller: Raspberry Pi, Arduino, or similar for processing and controlling the fountain.



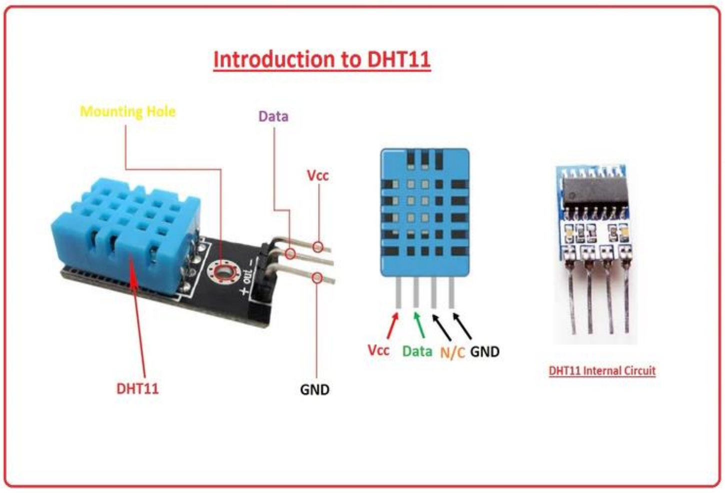


4. Internet Connectivity:Wi-Fi module or Ethernet connectivity for IoT communication.



5.Sensors (Optional):

Temperature and Humidity Sensor: To monitor environmental conditions.



Motion Sensor (PIR): To detect nearby people or animals and trigger the fountain.

6. Power Supply:Depending on the components, you might need a power source and potentially a backup system.

7. Control Interface:A user interface for control (could be a web interface, mobile app, or both).

<u>Steps to Create a Smart Water Fountain:</u>

<u>1. Design and Assemble:</u>

Design or modify your fountain to accommodate the electronics without compromising its aesthetic appeal or functionality. Install the water level sensor in the fountain basin.

<u>2. Sensor Integration:</u>

Connect the water level sensor to the microcontroller. This sensor will detect the water level and send this information to the controller.

<u>3. Microcontroller Setup:</u>

Program the microcontroller to read data from the water level sensor. This could involve setting thresholds for different water levels.

<u>4. IoT Integration:</u>

Connect the microcontroller to the internet using Wi-Fi or Ethernet. IoT platforms such as MQTT or cloud services like AWS IoT can be used for data transmission.

<u>5. Control and Monitoring Interface:</u>

Create a user interface for monitoring and controlling the fountain. This could be a web interface, mobile app, or both. Users can check water levels and remotely control the fountain (turning it on/off, adjusting flow, etc.).

6. Automation (Optional):

Implement automation. For instance, when the water level drops below a certain point, the system can automatically trigger a refill process or send alerts to the user.

7. Testing and Refinement:

Test the system thoroughly to ensure proper functionality. Refine and troubleshoot as necessary.

8. Maintenance and Updates:

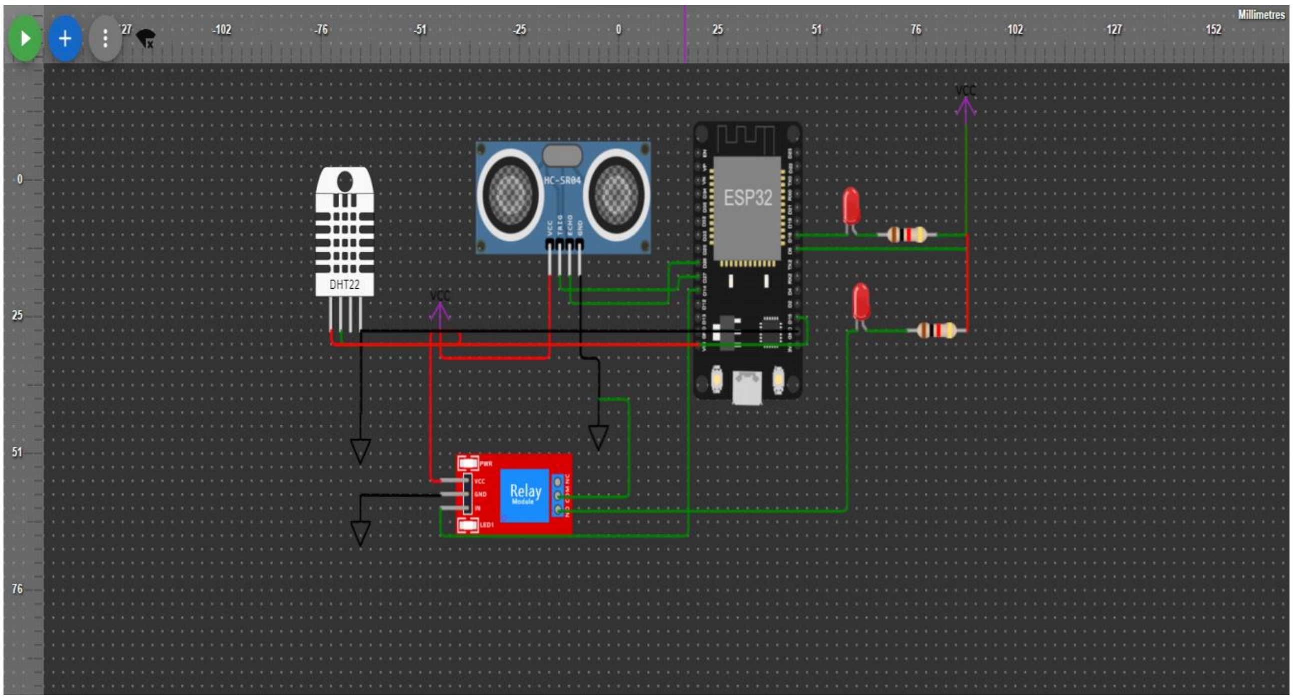
Regular maintenance and updates are crucial for ensuring the system's reliability and security.

Considerations:

Power Supply:	Ensure a reliable and stable power supply for continuous operation.
Security:	Implement security measures to prevent unauthorized access to the system.
Data Privacy:	Protect user data and privacy, especially if it involves any personal or sensitive information.

Creating a smart water fountain using IoT involves a combination of hardware, software, and IoT integration. It offers the convenience of remote monitoring and control while optimizing the fountain's functionality.

SCREENSHOT OF SIMULATION(OUTPUT):



CODE FOR SIMULATION:

```
#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);
}
```

SIMULATION LINK:

https://wokwi.com/projects/380214454467298305

WEB DEVELOPMENT CODE:

Creating a real-time water fountain status platform involves building a web application that can display and update data in real-time. To achieve this, you can use HTML, CSS, and JavaScript along with a backend server to provide the real-time data. Below is a simple example of how you can structure such a platform.

1. HTML (index.html)

```
&lt;!DOCTYPE html&gt;

&lt;html&gt;

&lt;head&gt;

&lt;title&gt;Water Fountain Status&lt;/title&gt;

&lt;link rel=&quot;stylesheet&quot; type=&quot;text/css&quot;
href=&quot;styles.css&quot;&gt;

&lt;/head&gt;

&lt;body&gt;

&lt;h1&gt;Water Fountain Status&lt;/h1&gt;

&lt;div id=&quot;status&quot;&gt;

&lt;p&gt;Flow Rate: &lt;span id=&quot;flowRate&quot;&gt;Updating...&lt;/span&gt; liters
per

minute&lt;/p&gt;

&lt;p&gt;Malfunction Alert: &lt;span
id=&quot;malfunction&quot;&gt;No&lt;/span&gt;&lt;/p&gt;

&lt;/div&gt;

&lt;script src=&quot;script.js&quot;&gt;&lt;/script&gt;

&lt;/body&gt;

&lt;/html&gt
```

2. CSS (styles.css)

```
body {
font-family: Arial, sans-serif;
text-align: center;
background-color: #f0f0f0;
}
h1 {
color: #333;
}
#status {
background-color: #fff;
padding: 20px;
border-radius: 10px;
box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
margin: 20px auto;
width: 300px;
}
#flowRate {
color: #009900;
font-weight: bold;
}
#malfunction {
color: #FF0000;
font-weight: bold;
}
```

<u>3. JavaScript (script.js)</u>

```
function updateStatus() {
// Simulate real-time data
const flowRate = Math.random() * 10 + 1; // Random flow rate
between 1 and 10 liters per minute
const isMalfunction = Math.random() &lt; 0.1; // 10% chance of
malfunction
document.getElementById(&quot;flowRate&quot;).textContent =
flowRate.toFixed(2);
document.getElementById(&quot;malfunction&quot;).textContent =
isMalfunction ? &quot;Yes&quot; : &quot;No&quot;;
// Update every 5 seconds (you can adjust the interval)
setTimeout(updateStatus, 5000);
}
// Initial update
updateStatus();
```

 This simple platform will display the water fountain&#39;s current status, including the water flow rate and whether there is a malfunction alert. It uses JavaScript to simulate real-time updates, but in a real application, you would replace the simulated data with actual data from your water fountain sensors or a backend server.

 To implement real-time data updates with actual sensor data, you would typically use technologies like WebSockets, AJAX, or Server-

Sent Events (SSE) to receive and update data from the server.

Additionally, you would need a backend server to collect and provide

the data to the front-end. The choice of backend technology can

depend on your specific requirements and constraints.

1. Flow Rate:

This is a simulated value representing a flow rate in Gallons

Per Minute (GPM). It is updated every 2 seconds with a random value

between 0 and 10, rounded to two decimal places (e.g., &quot;4.25 GPM&quot;).

2. Malfunction Status:

This is a simulated status that indicates whether there&#39;s a

malfunction or not. It&#39;s updated every 2 seconds with a 20% probability

of showing &quot;Yes&quot; (indicating a malfunction) and an 80% probability of

showing &quot;No&quot; (indicating no malfunction).

OUTPUT:(For javascript)

```
Flow Rate: 6.75 GPM

Malfunction Status: No
```

These values will change every 2 seconds as the `updateStatus` function

is called repeatedly. You can place these values within HTML elements

with the corresponding IDs (`&#39;flow-rate-value&#39;` and `&#39;malfunction-status&#39;`)to see the updates on your web page

Creating a smart water fountain using a Raspberry Pi involves several components such as sensors, a pump, and possibly a relay or motor controller. Below is a high-level overview of how you can design a basic diagram and code using Python to control a smart water fountain:

Hardware Components:

1. Raspberry Pi
2. Water pump
3. Water level sensor (to detect water levels)
4. Relay or motor controller (if the pump requires higher power than the Pi can provide)

Circuit Diagram:

The circuit may vary based on the specific components you're using. Here's a basic representation:

```
Raspberry Pi GPIO Pins --- Relay/Motor Controller --- Water Pump
        |                   |
        |                   |
        |                   |
    Water Level Sensor ------------ GND
```

Raspberry Pi Data Transmission:

Let's say you have a Raspberry Pi that collects temperature and humidity data from a sensor and then transmits it to a computer or mobile device. Below is a hypothetical example output of such data transmission:

Raspberry Pi Python Script for Data Transmission:

```
import random

import time

# Simulating sensor data (temperature and humidity)

def read_sensor_data():

    temperature = round(random.uniform(20.0, 30.0), 2)

    humidity = round(random.uniform(40.0, 60.0), 2)

    return temperature, humidity

# Simulating data transmission

while True:

    temperature, humidity = read_sensor_data()

    # Here you can use different methods like MQTT, HTTP requests, etc. to transmit data

    # For example, print the data for demonstration purposes

    print(f"Temperature: {temperature}°C, Humidity: {humidity}%")

    time.sleep(5)  # Simulating 5 seconds delay between readings
```

This script generates simulated temperature and humidity data and prints it to the console, mimicking the transmission of data to an external system or application.

Mobile App UI Example

Designing a mobile app UI for displaying this data could involve creating screens to visualize the information received from the Raspberry Pi. Here's a simplified textual representation of a potential UI:

Temperature and Humidity Display in a Mobile App:

```
------------------------------------

|        Current Environment       |

------------------------------------

|  Temperature:   24.15°C          |

|  Humidity:      53.78%           |

------------------------------------

|       Last Updated: 11:30 AM     |

------------------------------------
```

This UI could be part of a mobile application where the current temperature and humidity data received from the Raspberry Pi would be displayed. The numbers would update in real-time or at set intervals as new data is transmitted.

The actual mobile app design and development would involve utilizing appropriate programming languages (like Swift for iOS or Java/Kotlin for Android), UI/UX design tools, and integrating the data reception logic to display the real-time information from the Raspberry Pi.

Please note that actual implementation and design might vary based on your specific requirements and the programming languages or frameworks you choose to use for the mobile app and Raspberry Pi communication.

<u>**CONCLUSION:**</u>

A smart water fountain utilizing IoT technology represents a remarkable advancement in modern infrastructure, offering a range of benefits that cater to both efficiency and user experience. Through the integration of IoT, this innovative fountain ensures enhanced functionality, real-time monitoring, and improved sustainability.

In conclusion, the implementation of IoT in the development of a smart water fountain brings forth an array of advantages, including efficient water usage, remote monitoring, improved user experience, and the potential for wider application in smart city initiatives. This innovation stands as a testament to the power of technology in revolutionizing our infrastructure towards a more sustainable and interconnected future.