|  |  |
| --- | --- |
|  | SMART WATER FOUNDATION |
| Smart Water Fountain  IOT  Problem Statement:  Lack of efficient monitoring and control systems for water fountains.  Problem Definition:  Traditional water fountains lack intelligent features, making it difficult to manage their performance, water consumption, and maintenance. With the advancement of IoT technologies, there is a need for a standalone device equipped with IoT capabilities to monitor and control water fountains efficiently.  Design Thinking:  Related Work  In recent years, there have been several notable projects and technologies related to smart water management systems and IoT-based devices. These initiatives have laid the foundation for the development of smart water fountains. Here are a few relevant examples:  Smart Irrigation Systems: Smart irrigation systems have been widely adopted in agriculture to optimize water usage. These systems utilize IoT sensors to collect data on soil moisture, weather conditions, and plant requirements. By analyzing this data, the irrigation systems can deliver precise amounts of water, reducing waste and ensuring optimal plant growth.  Smart City Water Management: Many cities have implemented IoT-based water management systems to address water scarcity and improve resource efficiency. These systems monitor water distribution networks, detect leaks in pipelines, and optimize water usage in various municipal applications. The knowledge and experience gained from these projects can be applied to the design of smart water fountains.  Water Quality Monitoring: IoT technologies have been employed in water quality monitoring systems to ensure the safety and purity of water sources. These systems utilize sensors to measure parameters such as pH levels, turbidity, and chemical contaminants. By integrating similar sensors into smart water fountains, it becomes possible to monitor and maintain the quality of the water being used.  Smart Home Automation: The concept of smart homes has gained popularity, with IoT devices being used to automate various household tasks. Smart home automation systems can be extended to include water fountains, allowing users to control fountain settings, water flow, and lighting through voice commands or mobile applications. Integration with existing smart home ecosystems can enhance the user experience.  By studying and drawing insights from these related works, we can leverage their technological advancements, design principles, and best practices to develop an efficient and intelligent smart water fountain system.  Approach:  This section discusses the design and implementation of a smart water fountain. It includes the selection of appropriate IoT sensors and actuators, integration of these components into a standalone device, and the incorporation of machine learning algorithms for optimization.  System Overview:  +-----------------+  | User Interface|  +--------+--------+  |  +--------v--------+  |Central Control |  | Unit |  +--------+--------+  |  |  +---------------v-----------------+  | |  | Smart Water Fountain |  | |  +---------------+-----------------+  |  |  +-------------------v-------------------+  | |  | IoT Sensors and Actuators |  | |  +-------------------+-------------------+  |  |  +-------v--------+  | Cloud Database | +-----------------+  The smart water fountain comprises a network of IoT sensors, actuators, and a central control unit. The sensors collect data on water levels, temperature, and other parameters, while the actuators control water flow, lighting, and other fountain features. The central control unit processes the sensor data, communicates with the user interface, and manages the overall operation of the fountain.  Conclusion:  In conclusion, the development of a smart water fountain using IoT technologies offers numerous benefits, including efficient monitoring, control, and optimization of water consumption. This document highlights the potential of a smart water fountain to revolutionize the management of water fountains, making them more sustainable and environmentally friendly.  smart water fountain project:   1. Project Planning and Requirements Gathering:    * Define the goals of your project, such as promoting water conservation, ensuring water quality, or providing real-time data on water usage.    * Identify the specific parameters you want to monitor, such as water flow rate, temperature, pH levels, or water volume. 2. Hardware Selection:    * Choose the necessary IoT hardware components, including flow sensors, temperature sensors, pH sensors, microcontrollers, communication modules, and power sources.    * Select sensors based on the parameters you plan to monitor, ensuring they are suitable for water applications. 3. IoT Platform Selection:    * Select an IoT platform or framework to manage data collection, storage, and device management.    * Consider options like AWS IoT, Google Cloud IoT, Microsoft Azure IoT, or open-source platforms like MQTT. 4. Sensor Deployment:    * Install sensors in the smart water fountain, considering their placement for accurate monitoring and avoiding interference with the fountain's functionality.    * Calibrate and configure sensors as needed to ensure accurate data collection. 5. Data Collection and Connectivity:    * Configure sensors to collect data at regular intervals or in response to specific events, such as changes in water flow or temperature.    * Use wireless communication protocols like Wi-Fi, Bluetooth, or LoRaWAN to transmit data from the sensors to the IoT platform. 6. Data Processing and Analysis:    * Utilize the IoT platform to process and analyze the data collected from the sensors.    * Implement algorithms to detect anomalies, trends, or events related to water parameters, such as detecting leaks or unusual water consumption patterns. 7. User Interface:    * Develop a user interface, such as a web-based dashboard or a mobile app, to visualize the data collected from the smart water fountain.    * Provide real-time information on water flow, temperature, pH levels, and other relevant parameters.    * Include features like historical data, water usage statistics, and alerts for low water levels or abnormal conditions. 8. Alerting and Notifications:    * Set up alerting mechanisms to notify relevant personnel or users when predefined thresholds are exceeded or when anomalies occur, such as detecting a leak or water contamination. 9. Data Storage and Archiving:    * Store historical data in a secure and scalable database to enable long-term analysis and reporting.    * Implement data retention policies based on your project's requirements, considering factors like data storage capacity and compliance regulations.   10.Security and Privacy:   * + Ensure that the data collected from the smart water fountain is secure and complies with privacy regulations.   + Implement encryption, authentication, and access control measures to protect the data and prevent unauthorized access.   11.Energy Efficiency:   * + Optimize power management to extend the lifespan of battery-powered devices, such as using low-power sensors and sleep modes.   + Consider energy-efficient features like solar panels or power-saving configurations to minimize energy consumption.   12.Testing and Quality Assurance:   * + Thoroughly test the entire system to ensure accurate data collection, device reliability, and proper functioning of the IoT platform.   + Conduct field tests to validate the system's performance in real-world conditions, simulating various water usage scenarios.   13.Deployment:   * + Deploy the smart water fountain monitoring system in the desired location, ensuring proper installation, configuration, and connectivity.   + Test the system after deployment to verify its functionality and data accuracy.  1. Monitoring and Maintenance:    * Continuously monitor the system's performance and address any issues or malfunctions promptly.    * Perform regular maintenance tasks, such as sensor calibration, firmware updates, and cleaning of the water fountain components. 2. Data Analytics and Reporting:    * Analyze the collected data to gain insights into water usage patterns, identify opportunities for conservation, and detect any abnormalities.    * Generate reports and visualizations to inform decision-making and support water management efforts. |
|
|
|
|
|
|
|
|
|  |

Smart water fountain

"Our smart water fountain incorporates advanced sensors to conserve water by automatically turning off when no one is present."

"With personalized design options, our innovative fountain creates a unique experience for users while promoting water conservation."

"Using presence detection technology, the fountain intelligently turns on when someone approaches, providing water only when needed."

"By integrating touch-sensitive controls, users can easily activate and deactivate the fountain, ensuring water is not wasted."

"Our smart water fountain combines technology and sustainability, making it an eco-friendly solution for public spaces, offices, and homes."

Code:

/\*

PIR sensor tester

\*/

int ledPin = 12; // choose the pin for the LED int inputPin = 23; // choose the input pin (for PIR sensor) int pirState = LOW; // we start, assuming no motion detected int val = 0; // variable for reading the pin status void setup() { pinMode(ledPin, OUTPUT); // declare LED as output pinMode(inputPin, INPUT); // declare sensor as input

**Serial**.begin(9600);

} void loop() { val = digitalRead(inputPin); // read input value if (val == HIGH) { // check if the input is HIGH digitalWrite(ledPin, HIGH); // turn LED ON

if (pirState == LOW) { // we have just turned on

**Serial**.println("Motion detected!");

// We only want to print on the output change, not state pirState = HIGH;

}

} else {

digitalWrite(ledPin, LOW); // turn LED OFF

if (pirState == HIGH) { // we have just turned of

**Serial**.println("Motion ended!");

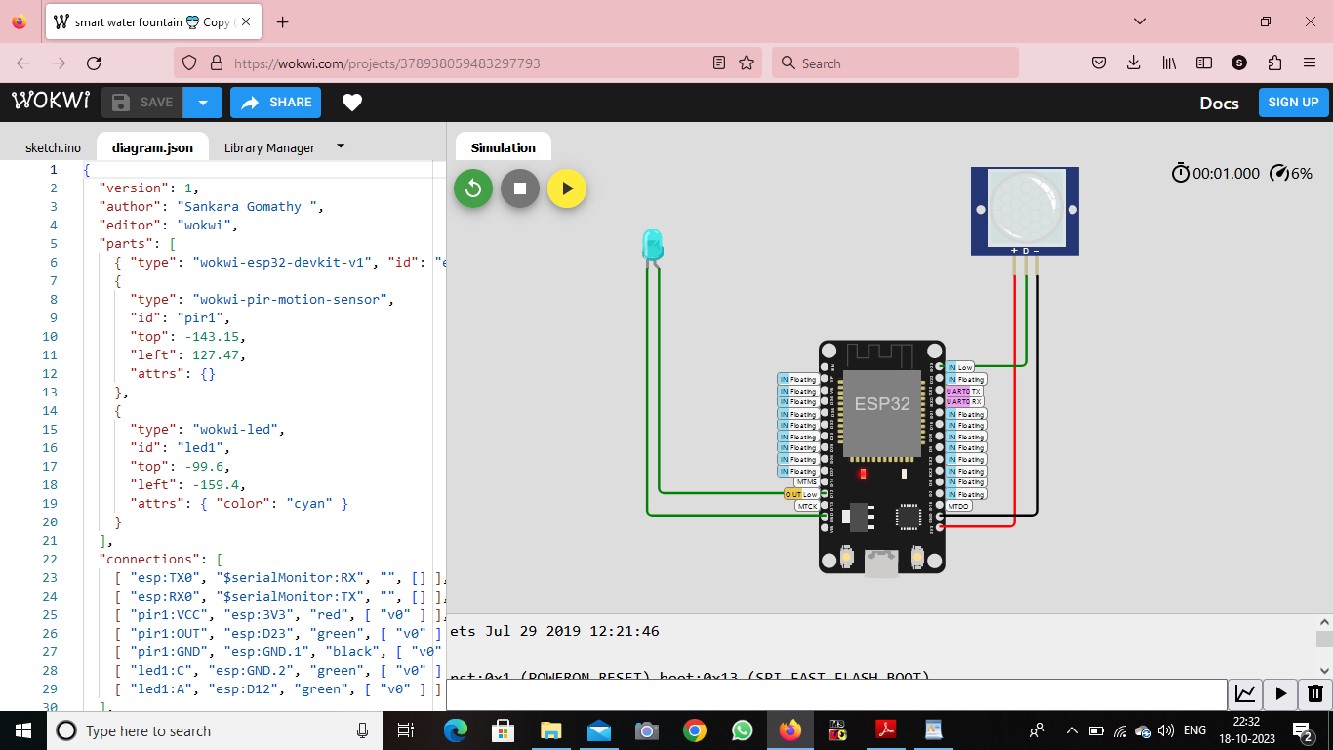
// We only want to print on the output change, not state pirState = LOW;

}

}

}

Stimulation:



Used component:

>ESP32

>pir sensor

>bulb represent the water fountain

Pin Assignments:

The pin assignments are defined using const int variables to specify the pin numbers for the presence sensor, touch sensor, and fountain relay. Adjust these values to match your specific hardware connections.

Variables:

Two boolean variables are used: is Person Present and is Fountain On. These flags track the presence of a person and the state of the fountain, respectively.

Setup:

In the setup() function, the pin modes are set for the presence sensor (input), touch sensor (input with pull-up resistor), and fountain relay (output).

LINK:

<https://wokwi.com/projects/378938059483297793>

SMART WATER FOUNDATION

**Wokwi-project:**

**Downloaded from**

[**https://wokwi.com/projects/378938059**](https://wokwi.com/projects/378938059483297793)

[**483297793**](https://wokwi.com/projects/378938059483297793)

**Simulate this project on** [**http://wokwi.com**](http://wokwi.com/)

Sketch.ino

// Pin assignments

const int pirSensorPin = 23; // Pin connected to PIR motion sensor

const int ultrasonicTriggerPin = 33; // Pin connected to Ultrasonic sensor trigger const int ultrasonicEchoPin = 12; // Pin connected to Ultrasonic sensor echo const int switchRelayPin = 26; int pirState = LOW;

int val = 0; // Pin connected to the

switch relay

// Variables

bool isMotionDetected = false; // Flag to track motion detection

void setup() { // Initialize pins

pinMode(pirSensorPin, INPUT); pinMode(ultrasonicTriggerPin, OUTPUT);

pinMode(ultrasonicEchoPin, INPUT); pinMode(switchRelayPin, OUTPUT); digitalWrite(switchRelayPin, LOW); // Turn off the switch initially // Other setup code for Ultrasonic sensor if needed

}

void loop() {

// Check PIR motion sensor val = digitalRead(pirSensorPin); // read input value

if (val == HIGH) { // check if the input is HIGH

digitalWrite(switchRelayPin, HIGH);

// turn LED ON

if (pirState == LOW) { // we have just turned on

Serial.println("Motion detected!"); // We only want to print on the

output change, not state

pirState = HIGH;

}

} else {

digitalWrite(switchRelayPin, LOW); //

turn LED OFF

if (pirState == HIGH) { // we have just turned of

Serial.println("Motion ended!"); // We only want to print on the

output change, not state pirState = LOW;

}

}

// Check Ultrasonic sensor int distance = measureDistance();

// Control switch based on motion and distance

if (isMotionDetected || distance <=

100) {

digitalWrite(switchRelayPin, LOW);

// Turn on the switch

} else {

digitalWrite(switchRelayPin, HIGH);

// Turn off the switch

}

}

int measureDistance() {

// Send a pulse to the Ultrasonic sensor digitalWrite(ultrasonicTriggerPin, LOW);

delayMicroseconds(2);

digitalWrite(ultrasonicTriggerPin, HIGH);

delayMicroseconds(10); digitalWrite(ultrasonicTriggerPin,

LOW);

// Measure the duration of the pulse long duration =

pulseIn(ultrasonicEchoPin, HIGH);

// Calculate the distance based on the

speed of sound

// (343 m/s or 0.0343 cm/µs)

int distance = duration \* 0.0343 / 2; return distance;

}

Diagram.json:

**{**

**"version": 1,**

**"author": "sankara gomathy ",**

**"editor": "wokwi",**

**"parts": [**

**{ "type": "wokwi-esp32-devkit-v1", "id": "esp", "top": -110.5, "left": -14.6,**

**"attrs": {} },**

**{**

**"type": "wokwi-pir-motion-sensor",**

**"id": "pir1",**

**"top": -216.8,**

**"left": 194.22,**

**"attrs": {}**

**},**

**{**

**"type": "wokwi-led",**

**"id": "led1",**

**"top": 111.6,**

**"left": -284.2,**

**"attrs": { "color": "cyan" }**

**},**

**{ "type": "wokwi-hc-sr04", "id": "ultrasonic1", "top": -228.9, "left": -**

**330.5, "attrs": {} },**

**{ "type": "wokwi-slide-switch", "id": "sw1", "top": 129.2, "left": 281.5,**

**"attrs": {} }**

**],**

**"connections": [**

**[ "esp:TX0", "$serialMonitor:RX", "", [] ],**

**[ "esp:RX0", "$serialMonitor:TX", "", [] ],**

**[ "ultrasonic1:VCC", "esp:VIN", "red", [ "v0" ] ],**

**[ "ultrasonic1:TRIG", "esp:D33", "green", [ "v0" ] ],**

|  |
| --- |
| **[ "ultrasonic1:ECHO", "esp:D12", "green", [ "v0" ] ],**  **[ "ultrasonic1:GND", "esp:GND.2", "black", [ "v0" ] ],**  **[ "led1:C", "esp:GND.2", "green", [ "v0" ] ],**  **[ "led1:A", "esp:D26", "green", [ "v0" ] ],**  **[ "pir1:VCC", "esp:3V3", "red", [ "v163.2", "h0", "v-86.4" ] ],**  **[ "pir1:OUT", "esp:D23", "green", [ "v38.4", "h-153.74" ] ],**  **[ "pir1:GND", "esp:GND.1", "black", [ "v0" ] ],**  **[ "sw1:1", "esp:GND.1", "green", [ "v0" ] ],**  **[ "sw1:2", "esp:D4", "green", [ "v0" ] ] ],**  **"dependencies": {}**  **}** |

Stimulation:

