SMART WATER FOUNTAIN

Deploy IoT sensors such as flow rate sensors, pressure sensors) in public water fountains to monitor water flow and detect malfunctions .To Develop a Python script on the IoT sensors to send real-time water fountain status data to the platform.

Building an IoT-enabled Smart Water Fountains system involves several steps, from setting up IoT sensors to developing a Python script for data transmission. Below is a high-level guide to help you get started:

Introduction:

The development of IoT-enabled Smart Water Fountains is essential for efficient management and maintenance of public water fountains. This paper outlines the process of assembling the required hardware, selecting an appropriate IoT platform, and implementing a Python script for data collection and transmission.

Components Needed:

1. ESP32 Development Board

2. Flow Sensor (e.g., YF-S201 or similar)

3. Pressure Sensor (e.g., BMP280 or similar)

4. Submersible Water Pump

5. Relay Module

6. 5V Power Supply for the ESP32

7. Tubing and Connectors

8. Breadboard and Jumper Wires

9. A Water Container (to act as a miniature fountain)

1. Hardware Setup:

1.1 Flow Sensor Flow sensors are critical components for measuring fluid flow. In this project, the YF-S201 flow sensor is chosen for its reliability and accuracy. The flow sensor's pulse output is connected to one of the ESP32's GPIO pins.

1.2 Pressure Sensor The BMP280 pressure sensor is employed to monitor water pressure within the fountain. The I2C interface (SCL and SDA pins) is used to connect the pressure sensor to the ESP32, ensuring precise pressure measurements.

1.3 Water Pump and Relay Module A submersible water pump is utilized to maintain water circulation within the fountain. The pump is connected to a relay module, which, in turn, is controlled by one of the ESP32's GPIO pins. This setup enables automated control of the water pump.

- Choose appropriate IoT sensors such as flow rate sensors and pressure sensors that can monitor water flow and detect malfunctions in public water fountains.
- Connect the selected sensors to a microcontroller or IoT development board. Popular choices include, Arduino, or ESP32.
- Ensure the sensors are securely placed within the water fountains for accurate data collection.

2. IoT Platform Selection:

- Select an IoT platform to collect and manage the sensor data. Popular choices include AWS IoT, Google Cloud IoT, Microsoft Azure IoT, or platforms like ThingSpeak , Ubidots, or Adafruit IO.
- The selection of an appropriate IoT platform is a pivotal decision in the development of an IoT-enabled Smart Water Fountain system. The chosen platform will serve as the backbone for efficiently collecting, processing, and managing data from the system's sensors. This section delves into the critical aspects of IoT platform selection, emphasizing the importance of robust data handling, security, and analytics capabilities.

3. Python Script for Data Collection:

- Develop a Python script to read data from the connected sensors. Depending on the sensor types, you may need to install appropriate Python libraries.
- Use GPIO pins or analog-to-digital converters to interface with the sensors.

4. Data Transmission to IoT Platform:

Use the platform-specific libraries or SDKs to send data to your chosen IoT platform. You'll need to set up device authentication and security.

5. Data Analysis and Visualization:

- Once data is sent to the IoT platform, you can set up data analytics and visualization tools to monitor the water fountains' status in real-time.
- Create dashboards, alerts, and reports to keep track of water flow and detect malfunctions.

6.Maintenance and Monitoring

Regular maintenance and monitoring of the system are essential for ensuring its continued functionality. This includes:

- Monitoring sensor accuracy and calibrating them as necessary.
- Verifying the connectivity of the system and addressing any connection issues.
- Implementing automated alerts or notifications for critical situations, such as a sudden drop in water flow or sensor malfunctions.
- Conducting routine inspections of the fountain's physical components, including the water pump and tubing.

FLOW SENSOR:

A flow sensor is a component that measures the flow of a fluid such as a gas or liquid. Flow sensors utilize both mechanical and electrical subsystems to measure changes in the fluid's physical attributes and calculate its flow.

ESP32 :

ESP32 stands for Espressif32 which is a development board developed by Espressif Systems. ESP32 is a 32-bit microcontroller equipped with a wireless or wifi network and Bluetooth Low energy (BLE) using the 802.11 b/g/n wifi network protocol that works at a frequency of 2.4 GHz and bluetooth v4.

PRESSURE SENSOR:

A pressure sensor is a device that senses and measures pressure. In this case, pressure is defined as the amount of force exerted over an area. Pressure sensors allow for more specialized maintenance strategies, such as predictive maintenance. These devices collect real-time data on the conditions of Equipment.

It Describes Each Hardware And Software Component Used In our Smart Water Fountain Project:

Component	Description
Hardware Components	
ESP32 Development Board	A 32-bit microcontroller with Wi-Fi and Bluetooth capabilities, serving as the project's core controller.
Flow Sensor (YF-S201)	A flow sensor used to measure the rate of water flow through the fountain, equipped with a pulse output for data collection.
Pressure Sensor (BMP280)	A pressure sensor that measures water pressure within the fountain. It communicates using the I2C interface, providing real-time pressure data.
Submersible Water Pump	A water pump responsible for circulating water within the fountain. It maintains water flow as per system requirements.
Relay Module	A module that controls the water pump's operation. It's used to turn the pump on or off based on system requirements.
5V Power Supply	A power supply providing a stable 5V power source for the ESP32, ensuring consistent operation.
Tubing and Connectors	Tubing and connectors are used for water circulation within the system, connecting the pump, flow sensor, and water container.
Water Container	A container acting as a miniature fountain, where water flow

	and pressure are monitored and controlled.
Breadboard and Jumper Wires	Breadboards and jumper wires facilitate electrical connections and prototyping.
Software Components	
Python Script	A Python script developed to run on the ESP32, responsible for collecting data from the sensors, managing sensor data, and transmitting it to the selected IoT platform.
IoT Platform	The chosen IoT platform (e.g., AWS IoT, Google Cloud IoT, ThingSpeak) responsible for collecting, storing, and analyzing sensor data, providing remote access and management of the system.
Sensor Libraries	Specific Python libraries used to interface with the flow sensor and pressure sensor to ensure data accuracy and reliability.
IDE (Integrated Development Environment)	An IDE, such as Arduino IDE or Thonny, is used for writing, compiling, and uploading the Python script to the ESP32 microcontroller.

Project Steps:

1. Assemble the Hardware:

- Set up your miniature fountain using the water container, water pump, and tubing. Ensure that the water pump is securely placed in the container.
- Connect the flow sensor to the water circulation system. The flow sensor typically has a pulse output. Connect its output to a GPIO pin on the ESP32.
- Install the pressure sensor at a suitable location in the water container to monitor water pressure. Connect it to the ESP32 using the I2C interface (SCL and SDA pins).
- Connect the water pump to the relay module and ensure that it's properly connected to one of the ESP32's GPIO pins.

2. Writing the Firmware for the IoT-Enabled Smart Water Fountain System:

Develop firmware to control the water fountain. Use the Arduino IDE or Platform IO for ESP32 development.

1. **Sensor Data Integration:** The firmware development process involves integrating the sensors (flow rate and pressure sensors) with the ESP32 microcontroller. This integration requires configuring GPIO pins and communication protocols (such as I2C) for accurate data collection. The firmware should read data from these sensors efficiently and at regular intervals to provide real-time information about water flow and pressure.
2. **Data Processing and Transmission:** In the firmware, data collected from the sensors should be processed, organized, and prepared for transmission to the chosen IoT platform. This step involves applying any necessary calibration or data conversion to ensure the transmitted data is accurate and consistent. The firmware should also handle the secure transmission of data using protocols like MQTT or HTTP to the IoT platform.
3. **Device Control:** The firmware should provide mechanisms to control the water pump based on the sensor data. It should include logic to adjust the pump's speed or activation based on changes in water pressure or other predetermined conditions. This control logic ensures that the water fountain responds appropriately to maintain the desired water flow rate.
4. **Error Handling and Diagnostics:** Robust firmware should incorporate error handling mechanisms to detect and report any anomalies or malfunctions in the system. It should provide diagnostic capabilities, allowing administrators to identify issues promptly. Error logs or notifications can be part of the firmware to help with system maintenance and troubleshooting.

PROGRAM:

```
///THE SMART WATER FOUNTAIN USING IOT Project
#define PIN_TRIG 26
#define PIN_ECHO 25
#define LOWLED 18
#define MIDLED 19
#define HIGHLED 21
#define MOTOR 27
unsigned int level = 0;
```

```
void setup() {

  pinMode(LOWLED, OUTPUT);
  pinMode(MIDLED, OUTPUT);
  pinMode(HIGHLED, OUTPUT);
  pinMode(MOTOR, OUTPUT);
  digitalWrite(LOWLED, HIGH);
  digitalWrite(MIDLED, HIGH);
  digitalWrite(HIGHLED, HIGH);
  digitalWrite(MOTOR, LOW);

  Serial.begin(115200);
  pinMode(PIN_TRIG, OUTPUT);
  pinMode(PIN_ECHO, INPUT);
}999999999

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(LOWLED, LOW);
    digitalWrite(MOTOR, HIGH);
    digitalWrite(HIGHLED, HIGH);
    digitalWrite(MIDLED, HIGH);
  }

  else if ((level > 200 ) && (level < 400))
  {
    digitalWrite(LOWLED, HIGH);
    digitalWrite(HIGHLED, HIGH);
    digitalWrite(MIDLED, LOW);
  }
```

```
 else if (level >= 400 )
 {
  digitalWrite(HIGHLED, LOW);
  digitalWrite(MIDLED, HIGH);
  digitalWrite(LOWLED, HIGH);
  digitalWrite(MOTOR, LOW);
 }
 delay(1000);
}
```

CODE EXPLANATION:

The Above Arduino code is designed for an ultrasonic sensor-based system that measures distance and controls LEDs and a motor based on the detected distance. It uses an ultrasonic sensor to measure the distance of an object in front of it and displays this distance in both centimeters and inches. Additionally, it controls three LEDs and a motor based on the measured distance.

- ❖ Here, the code defines the pin numbers for the ultrasonic sensor's trigger and echo pins (PIN_TRIG and PIN_ECHO) and pin numbers for three LEDs (LOWLED, MIDLED, HIGHLED) and a motor (MOTOR).
- ❖ It also declares a global variable **level** to store the distance measurement.

- ❖ In the **setup()** function, the code configures the pins for the LEDs and motor as OUTPUTs. It initializes all LEDs as initially turned on (HIGH) and the motor as initially turned off (LOW).
- ❖ The **Serial.begin(115200)** initializes serial communication for debugging. It also sets the trigger pin as an OUTPUT and the echo pin as an INPUT.

In the **loop()** function, the code continuously performs the following steps:

- It triggers the ultrasonic sensor by setting the trigger pin (PIN_TRIG) to HIGH and then LOW after a short delay.
- It uses **pulseIn()** to measure the time it takes for the sensor's echo pin (PIN_ECHO) to go HIGH after sending the trigger signal. This duration is proportional to the distance of the object.
- It converts and prints the measured distance in both centimeters and inches using the speed of sound (58 µs/cm and 148 µs/inch).
- Based on the measured **level**, it controls the LEDs and motor:
  - If the distance is less than 100 cm, it turns on the motor and turns off the LOWLED.
  - If the distance is between 200 and 400 cm, it turns off MIDLED and turns on LOWLED.

    - If the distance is greater than or equal to 400 cm, it turns off HIGHLED and turns on MIDLED, while also turning off the motor.
- It introduces a delay of 1 second to control the loop rate.

3. Test and Observe:

- ❖ Upload the firmware to your ESP32.
- ❖ Monitor the serial output to check the flow rate and pressure values.
- ❖ Observe the water fountain to see how the pump responds to changes in pressure.

1. **Firmware Upload:** Uploading the firmware to the ESP32 is the initial step in validating the IoT-enabled Smart Water Fountain system. It ensures that the microcontroller is equipped with the necessary code to collect and transmit sensor data.
2. **Serial Output Monitoring:** The ability to monitor the serial output from the ESP32 is essential for real-time validation. It enables the continuous observation of flow rate and pressure values, ensuring the sensors are providing accurate and up-to-date data.
3. **Water Pump Response:** Observing the water pump's response to changes in water pressure is a crucial test. This ensures that the system can adjust water flow according to varying conditions, maintaining an optimal and consistent water fountain experience.
4. **Data Validation:** The collected data is subjected to validation, ensuring that the sensor readings are precise and reliable. Any discrepancies or anomalies are identified, and appropriate actions are taken to rectify them.
5. **Fine-Tuning and Optimization:** The testing and observation phase offers valuable insights into the system's performance. It allows for the fine-tuning and optimization of the system, addressing any issues or inefficiencies to ensure that it functions effectively in real-world applications.

PROJECT SIMULATION LINK:

https://wokwi.com/projects/378837482644527105

CONCLUSION:

The implementation of the IoT-enabled Smart Water Fountain system, featuring robust flow and pressure sensors connected to the ESP32, demonstrates a remarkable advancement in public fountain management. This system offers real-time data collection, ensuring precise water flow control and rapid malfunction detection. Through careful platform selection, data handling, security, and analytics capabilities are optimized. Testing and observations validate the system's accuracy and responsiveness. The successful upload of firmware to the ESP32 and effective monitoring of serial output confirm the reliability of sensor data. Additionally, the water pump's responsiveness to pressure changes ensures a consistent and enjoyable experience. This project signifies a significant stride in enhancing public water fountain efficiency and maintenance.

TEAM MEMBERS:

NAME: ASHWIN KUMAR N A , REG NO:211521106016

NAME: MANISH KANDAN K , REG NO:211521106094

NAME: AVISALA LEELA MOHAN, REG NO:211521106018

NAME: KRITHIK KANVAR N, REG NO: 211521106090

NAME: DHANUSH KUMAR M S, REG NO: 211521106035