Smart Water Fountains

# 

**Akshay P**

**711121106001**

**Internet of Things**



# **INTRODUCTION**

Smart water fountains are a modern and technologically advanced approach to providing clean and convenient drinking water in public spaces. They offer a number of advantages over traditional water fountains, including:

* Improved user experience: Smart water fountains can be equipped with features such as touch-free operation, real-time water quality monitoring, and temperature control. This makes them more convenient and user-friendly than traditional water fountains.
* Sustainability: Smart water fountains can help to reduce water consumption and waste by using sensors to detect leaks and by monitoring usage patterns. This helps to conserve precious water resources and reduce environmental impact.
* Maintenance efficiency: Smart water fountains can be remotely monitored and managed, which can help to reduce maintenance costs and downtime. This is especially beneficial for high-traffic areas where water fountains need to be constantly operational.

This project aims to develop an IoT-based smart water fountain that can be used in public spaces. The water fountain will be equipped with a variety of sensors to monitor water quality, usage, and other factors. The sensor data will be transmitted to a cloud server, where it can be analyzed and used to control the water fountain and to provide real-time information to users.

The smart water fountain will be controlled using a Raspberry Pi. The Raspberry Pi will be responsible for reading the sensor data, controlling the water fountain, and communicating with the cloud server.

The smart water fountain will be equipped with a display to show the sensor data and to provide instructions to users. The water fountain will also have a speaker to play alerts, such as a notification if the water quality is low.

The smart water fountain will be powered by a solar panel and a battery. This will make the water fountain self-sufficient and independent of the electrical grid.

The smart water fountain will be designed to be easy to install and maintain. The water fountain will be modular and scalable, making it possible to customize it to meet the specific needs of different locations.

This project has the potential to make a significant contribution to improving the quality of public drinking water and to reducing water waste. The smart water fountain can also help to improve the sustainability and efficiency of public water infrastructure.

# **Project Goals**

* To develop a low-cost, easy-to-install, and easy-to-maintain smart water fountain.
* To develop a smart water fountain that can be used in public spaces to provide clean and convenient drinking water.
* To develop a smart water fountain that can monitor water quality, usage, and other factors.
* To develop a smart water fountain that can be controlled remotely via a cloud server.
* To develop a smart water fountain that can provide real-time information to users about water quality, usage, and other factors.
* To develop a smart water fountain that can help to reduce water consumption and waste.
* To develop a smart water fountain that can help to improve the sustainability and efficiency of public water infrastructure.

# **Project Activities**

**Fault Detection Sensor:**

Sensors used in the monitoring of machinery with the object of identifying fault.

**Flow Sensors:**

Flow sensors are placed in water distribution pipelines to measure water flow rates and detect any irregularities or leaks.

**Ultrasonic Sensors :**

For water level measurement and flow meter for flow rate measurement.

In addition to these activities, you may also want to consider the following:

* Develop a control system for the water fountain using a Raspberry Pi. The control system could be used to turn the water fountain on and off, to adjust the flow rate, and to monitor the sensor data.
* Develop a user interface for the water fountain. The user interface could be used to display the sensor data, to allow users to control the water fountain, and to provide alerts about water quality or other problems.
* Develop a cloud server to store and analyze the sensor data. The cloud server could also be used to control the water fountain remotely.
* Test the water fountain in a real-world environment. Once the water fountain is developed, it is important to test it in a real-world environment to ensure that it is reliable and easy to use.
* Develop a Python script to collect and process the data from the sensors in real time.

The following Python script was developed to collect and process the data from the sensors in real time:

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

# Import the necessary library

import tinkercad

import requests # Add the requests library for HTTP requests

# Initialize the

Tinkercad API

tc = tinkercad.Tinkercad(username="

vignesh03 ", vignesh

# Find the simulation you want to run (use the correct ID)

simulation\_id = "

2303774

# Get the simulation

simulation = tc.get\_simulation(simulation\_id)

# Define the ThingSpeak

API parameters

thingspeak\_api\_key = "

G5YN4PEKH3VEQ0JA

thingspeak\_url = f"https://api.thingspeak.com/update?api\_key={thingspeak\_api\_key}"

# Define the code to run in the Arduino

arduino\_code = """

#include <Ultrasonic.h>

Ultrasonic ultrasonic(2, 3); // T

rigger (pin 2), Echo (pin 3)

void setup() {

Serial.begin(9600);

}

void loop() {

float distance = ultrasonic.Ranging(CM);

Serial.println(distance);

// Send data to the computer (Python

Serial.print("D:");

Serial.println(distance);

delay(1000);

}

"""

"""

# Upload and run the code in the simulation

# Upload and run the code in the simulation

simulation.run\_code(arduino\_code)

simulation.run\_code(arduino\_code)

# Monitor the water level and send data to ThingSpeak

# Monitor the water level and send data to ThingSpeak

while True:

while True:

data = simulation.get\_serial\_data()data = simulation.get\_serial\_data()

if data and data.startswith("D:"):if data and data.startswith("D:"):

distancedistance = float(data[2:])= float(data[2:])

print(f"Water level: {distance} cm")

print(f"Water level: {distance} cm")

# Send data to ThingSpeak# Send data to ThingSpeak

try:try:

response = requests.get(f"{thingspeak\_url}&field1={distance}")response = requests.get(f"{thingspeak\_url}&field1={distance}")

if response.status\_code == 200:if response.status\_code == 200:

print("Data sent to ThingSpeak s

print("Data sent to ThingSpeak successfully.")uccessfully.")

else:else:

print("Failed to send data to ThingSpeak.")

print("Failed to send data to ThingSpeak.")

except Exception as e:except Exception as e:

print("Error sending data to ThingSpeak:", str(e))

print("Error sending data to ThingSpeak:", str(e))

These lines import the necessary libraries:

* time library is used to add delays to the code.
* board library provides information about the Raspberry Pi board that is being used.
* RPi.GPIO library provides functions for controlling the GPIO pins on the Raspberry Pi.
* requests library is used to make HTTP requests to the cloud server.

# Set the GPIO pin that is connected to the relay module

relay\_pin = 18

This line sets the GPIO pin that is connected to the relay module. In this case, the relay module is connected to GPIO pin 18.

# Initialize the GPIO pin

GPIO.setmode(GPIO.BCM)

GPIO.setup(relay\_pin, GPIO.OUT)

These lines initialize the GPIO pin. The GPIO.setmode() function sets the GPIO mode to BCM mode. The GPIO.setup() function sets the GPIO pin to output mode.

# Set the cloud server URL

cloud\_server\_url = "https://example.com/api/water\_fountain\_data"

This line sets the cloud server URL. The cloud server is the server that stores the data from the water fountain, such as the water quality and the flow rate.

# Get the data from the cloud server

response = requests.get(cloud\_server\_url)

This line makes a GET request to the cloud server to get the data from the water fountain.

# Check the response status code

if response.status\_code == 200:

This line checks the response status code to see if the request was successful. If the status code is 200, then the request was successful and the data from the water fountain is stored in the response object.

# The request was successful

data = response.json()

This line converts the data from the response object to JSON format. The JSON data is stored in the data object.

# Do something with the data, such as turning on the water fountain if the water quality is good

if data["water\_quality"] >= 90:

GPIO.output(relay\_pin, GPIO.HIGH)

else:

GPIO.output(relay\_pin, GPIO.LOW)

This code checks the water quality from the data object. If the water quality is greater than or equal to 90, then the code turns on the water fountain. Otherwise, the code turns off the water fountain.

else:

# The request failed

print("Error: {}".format(response.status\_code))

If the request to the cloud server failed, then this code will print an error message to the console.

# Wait for 10 seconds

time.sleep(10)

This line waits for 10 seconds before turning off the water fountain.

# Turn off the water fountain

GPIO.output(relay\_pin, GPIO.LOW)

This line turns off the water fountain.

This is a simple example of how to use a Raspberry Pi to control a water fountain using a relay module and data from a cloud server. You can modify the code to add additional features, such as:

* Read sensor data: You can add code to the script to read data from sensors, such as a water quality sensor or a flow sensor. You can then use the sensor data to control the water fountain or to send alerts to users.
* Remote control: You can add code to the script to allow the water fountain to be controlled remotely via a web application or a mobile app. This would allow users to turn the water fountain on and off, adjust the flow rate, or receive alerts about water quality or other problems.
* Data logging: You can add code to the script to log the sensor data to a file or to a cloud server. This data can then be analyzed to identify trends and patterns, which can be used to improve the operation of the water fountain.

# **RESULTS**

If you use the python script that you have provided, the water fountain will be turned on for 10 seconds and then turned off. The script will not be able to do anything else, such as monitoring water quality or controlling the flow rate of the water fountain.

However, the script can be used as a starting point for developing your own IoT-based smart water fountain project. You can modify the script to add additional features, such as:

* Reading sensor data: You can add code to the script to read data from sensors, such as a water quality sensor or a flow sensor. You can then use the sensor data to control the water fountain or to send alerts to users.
* Remote control: You can add code to the script to allow the water fountain to be controlled remotely via a cloud server or a mobile app. This would allow users to turn the water fountain on and off, adjust the flow rate, or receive alerts about water quality or other problems.
* Data logging: You can add code to the script to log the sensor data to a file or to a cloud server. This data can then be analyzed to identify trends and patterns, which can be used to improve the operation of the water fountain.

Once you have modified the script to add the desired features, you can deploy it on a Raspberry Pi and connect it to the water fountain. The Raspberry Pi will then be able to control the water fountain and monitor the sensor data. The sensor data can be used to improve the operation of the water fountain or to send alerts to users.

# **CONCLUSION**

In conclusion, the python script that you have provided is a simple example of how to use a Raspberry Pi to control a water fountain using a relay module. The script can be used as a starting point for developing your own IoT-based smart water fountain project. You can modify the script to add additional features, such as reading sensor data, remote control, and data logging.

IoT-based smart water fountains offer a number of advantages over traditional water fountains, including:

* Improved user experience: Smart water fountains can be equipped with features such as touch-free operation, real-time water quality monitoring, and temperature control. This makes them more convenient and user-friendly than traditional water fountains.
* Sustainability: Smart water fountains can help to reduce water consumption and waste by using sensors to detect leaks and by monitoring usage patterns. This helps to conserve precious water resources and reduce environmental impact.
* Maintenance efficiency: Smart water fountains can be remotely monitored and managed, which can help to reduce maintenance costs and downtime. This is especially beneficial for high-traffic areas where water fountains need to be constantly operational.

IoT-based smart water fountains have the potential to make a significant contribution to improving the quality of public drinking water and to reducing water waste. Smart water fountains can also help to improve the sustainability and efficiency of public water infrastructure.