

# Smart Water Fountains

## Project Documentation

Date	23-10-2023
Team ID	509
Project Name	Smart Water Fountains

### 1.Introduction

Optimizing the efficiency of smart water fountains is crucial to promote water conservation and responsible usage. In this document, we will outline a Internet of Things (IoT) project focused on maximizing the efficiency of smart water fountains. We'll define the problem statement, outline the steps involved, and discuss how an automated approach can benefit water management and sustainability efforts.

### 2.Problem Statement

It is hard for us to optimize the water fountain to minimize the water usage based on various factors. We going to develop a project that optimizes the water usage of smart water fountains based on various factors such as foot traffic, weather, and time of day.

### 3.Design Thinking Approach

Empathize:

Before diving into solving the problem, it's crucial to empathize with the users—water management authorities and environmentalists in this case. Understand their needs and how optimizing smart water fountains can benefit water conservation efforts

Actions:

- Conduct interviews with water management authorities to gather insights into their challenges in optimizing water usage.
- Analyse historical fountain usage and environmental data to identify trends and patterns related to water usage efficiency.

- Seek input from environmentalists to understand the factors that influence water conservation efforts.

Define:

Based on insights gathered during the empathize phase, define clear objectives and success criteria for the smart water fountain efficiency optimization project..

#### **4.Objectives:**

- Develop a model that optimizes water usage in smart water fountains.
- Provide recommendations to water management authorities on efficient water usage strategies.

#### **5.Ideate:**

Brainstorm potential approaches for optimizing water usage in smart fountains and explore creative ideas to achieve the defined objectives.

Actions:

1. Explore different regression algorithms suitable for water usage optimization.
2. Consider incorporating real-time weather data and foot traffic analysis as additional features.
3. Brainstorm ways to visualize and present the model's recommendations to water management authorities.

Prototype

Create a prototype of the water usage optimization model for smart water fountains and design initial recommendations for efficient water usage.

Action

1. Develop Python scripts/C scripts or notebooks for data preprocessing, for Connection with cloud.
2. Provide preliminary recommendations to water management authorities based on the prototype's insights.
3. Apply the water usage optimization project to smart water fountains in specific locations to guide water usage decisions.
4. Share predictions and recommendations with water management authorities.

5. Continuously monitor and improve the projects performance as more fountain usage data becomes available.

### Test

Evaluate the accuracy of the water usage optimization project and gather feedback from water management authorities and environmentalists.

### Implement

Once the prototype meets the defined objectives and receives positive feedback, proceed with full implementation.

### Iterate

Continuous improvement is essential. Gather feedback, analyze prediction results, and iterate on the model to enhance water usage optimization and conservation efforts.

## **6.Design and Innovation Strategies**

Innovation: Real time environmental data processing

The ESP32 is a popular and versatile microcontroller and Wi-Fi/Bluetooth module developed by Espressif Systems. It's known for its powerful processing capabilities, dual-core CPU, and built-in wireless connectivity, making it a top choice for IoT and embedded projects. The ESP32 features a 32-bit Tensilica Xtensa LX6 CPU, up to 520KB of SRAM, various GPIO pins, support for Wi-Fi 802.11 b/g/n, Bluetooth 4.2 and BLE (Bluetooth Low Energy), and a wide range of peripherals. It can be programmed using the Arduino IDE or the Espressif IDF, and it's widely used for IoT devices, home automation, wearables, and more, thanks to its small form factor and low power consumption. The ESP32 is favoured for its affordability, robust community support, and extensive features, making it an ideal choice for a wide range of embedded applications. This processor is best suitable for the Real time data processing and wireless connectivity in efficient cost.

### **. Components**

Sensor selection

- Ultrasonic- To monitor the water level in the fountain.
- Turbidity sensor (in future)- To monitor the quality of water used in fountain.

- Temperature sensor (in future)-To monitor the temperature of water
- Other components
- o Relay- To connect the microcontroller to the Water pump to actuate the water pump
  - o Water pump – To draws water into its housing, where an forces the water out through the pump's outflow fitting. In other word used to recirculate the water. It also used to fill the reservoir.
  - o Power supply- To make the components work. We need two power supply for the fountain 5v for microcontroller and 230 v for water pump
  - o Fountain- Tubing and fountain nozzle, water proof container required to make the water fountain

## **Cloud platform**

Innovation: Realtime data access Cloud computing in IoT enables centralized data storage, scalable processing, remote device management, and global accessibility, making it essential for managing and analysing IoT data efficiently and securely. We decided to use firebase because of its real time data transmission, user-friendly, robust security features.

## **Connectivity**

Innovation: Real time connectivity

IoT devices communicate through the internet, collecting data and enabling remote control

and monitoring, creating interconnected and smart systems for various applications

Wi-fi - We decided to use "Wireless Fidelity," which is a wireless technology that enables

devices to connect to the internet and communicate with each other without physical cables,

using radio waves over short distances. It's commonly used for wireless internet access,

networking, and IoT connectivity

## **Protocol**

Innovation: Used to communicate

Protocol is a set of rules used in communication

In esp32 we use

Wi-Fi (IEEE 802.11): ESP32 provides built-in Wi-Fi support for wireless internet connectivity and

communication with other Wi-Fi devices. It enables IoT devices to connect to networks, access the internet, and exchange data.

**Bluetooth and Bluetooth Low Energy (BLE):** The ESP32 includes Bluetooth and BLE capabilities for

wireless communication with smartphones, sensors, and peripherals. This enables seamless data

exchange and control between devices.

**Serial Communication (UART, SPI, I2C):** The ESP32 supports standard serial communication

protocols like UART, SPI, and I2C, allowing it to interface with a wide range of sensors, displays,

and other microcontrollers.

**HTTP and HTTPS:** ESP32 can communicate over HTTP and secure HTTPS protocols, facilitating

web-based data transfer and interactions with cloud services.

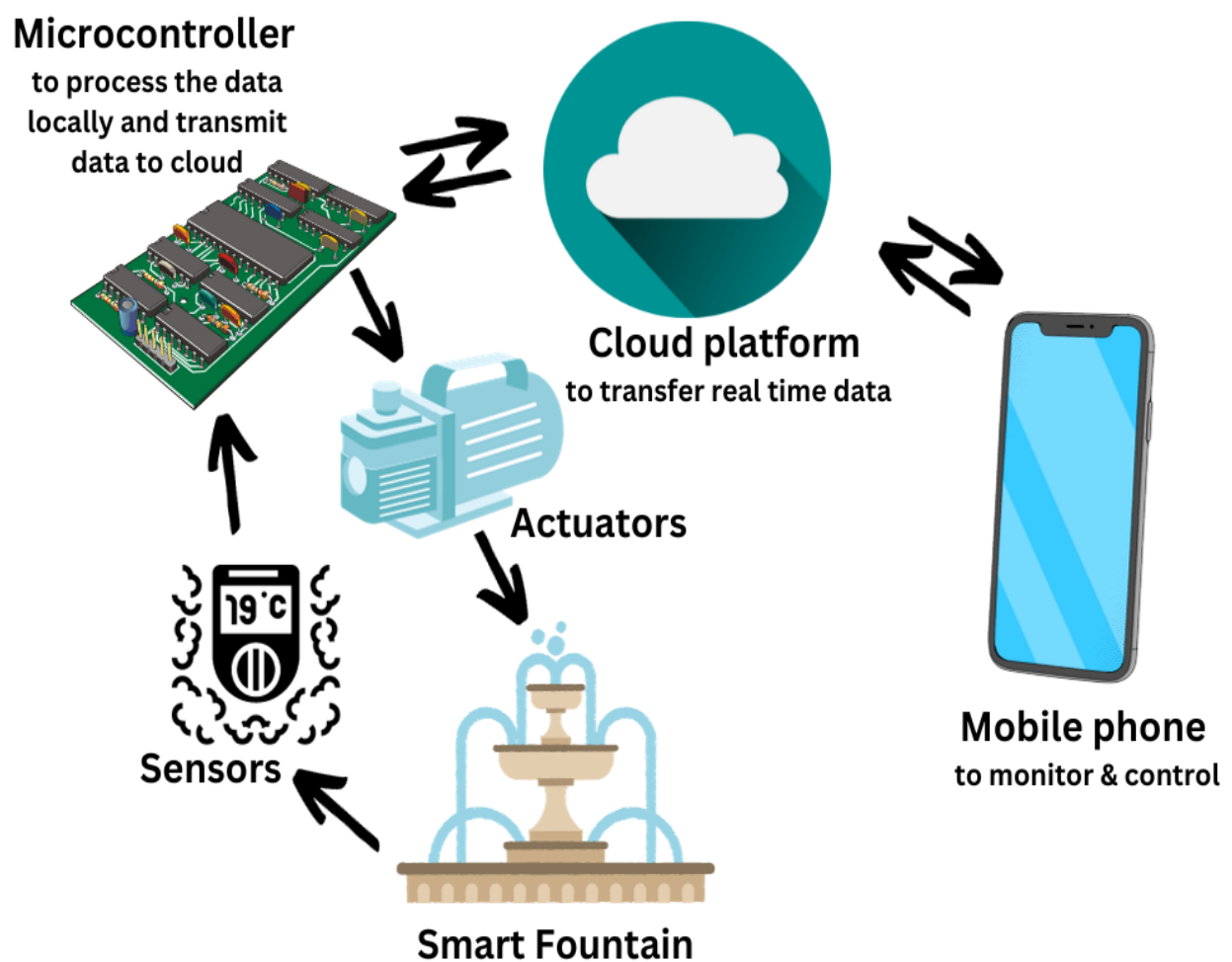
And Bluetooth, Low power Bluetooth etc

## **7.Use Case**

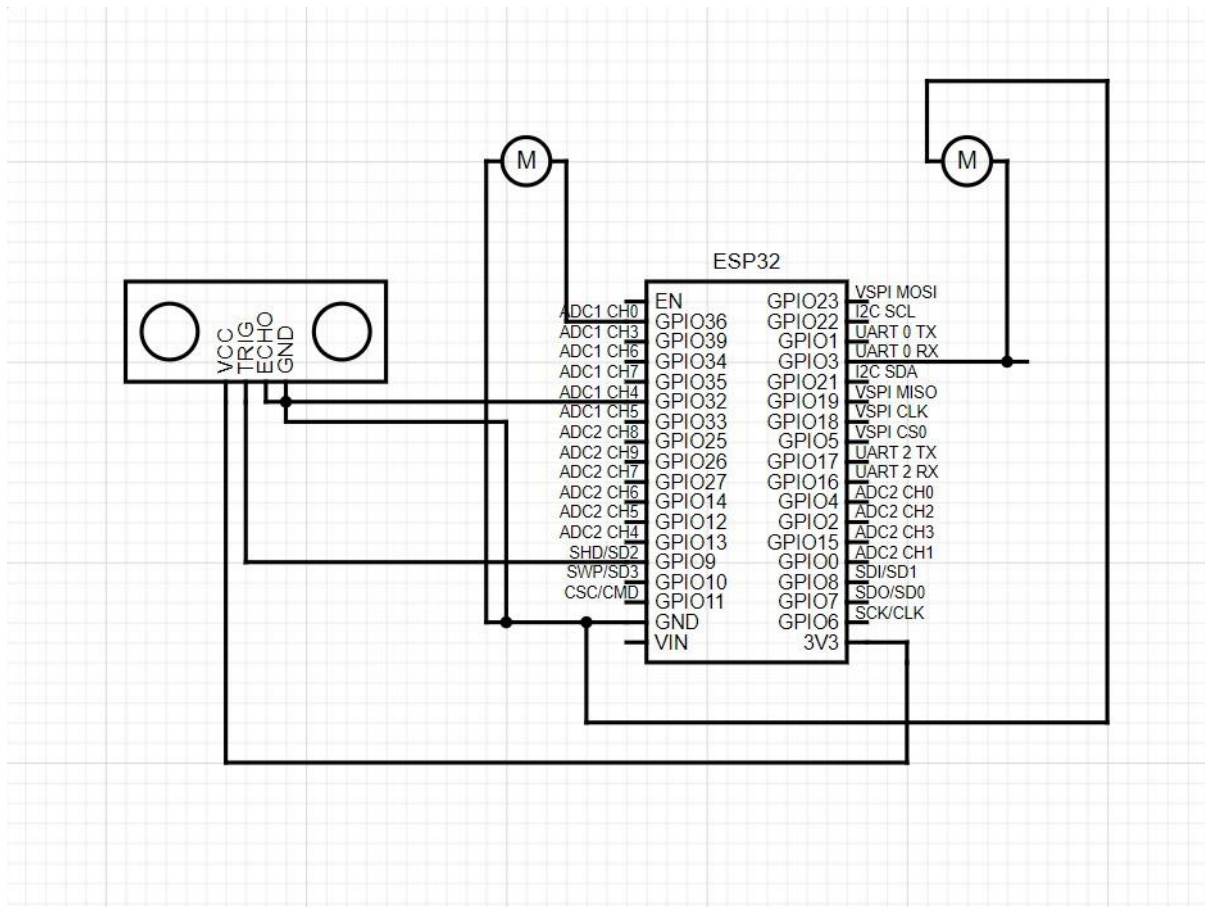
The fountain can monitor the quality of the water it dispenses, checking for impurities, pH levels, and contaminants. It can provide real-time data and alerts for maintenance. The fountain can maintain the water at a specific temperature range, ensuring it's comfortable for drinking or other intended uses. Implement features to prevent water wastage, such as shutting off the fountain after a certain duration or automatically adjusting flow based on demand. Allow administrators to remotely control and monitor the fountain's operation and status through a web or mobile application. Record data on fountain usage, water quality, and maintenance needs. Generate reports for maintenance scheduling and water quality analysis. Provide reminders for when water filters need replacement to maintain water quality. Implement energy-saving features, such as sensors to turn off the fountain when not in use or LED lighting for better visibility. In eco-friendly projects, the fountain could include water

recycling and purification systems to minimize water waste. Gather and analyse data on how much water is being consumed at different times and locations to optimize water distribution.

## 8.Block Diagram



## Schematic Diagram



## 9.Code

Since the Python Libraries are not available in Wokwi we did it Using C/C++

---

### Micro Python Code

```
from machine import Pin, PWM, Timer, ADC
from ultra import DistanceSensor
from time import sleep

ds = DistanceSensor(echo=14, trigger=15)

Fountain_motor = 2
motor_num=27

pin = machine.Pin(Fountain_motor, machine.Pin.OUT)
motor=machine.Pin(motor_num,machine.Pin.OUT)
while True:
```

```

distance_cm = ds.distance * 100
distance=float(distance_cm)
print(f"Distance: {distance_cm} cm")
#to refill water in the reservoir automatically-----
if distance < 10:
    #denotes the distance btw sensor and water
    print("full")
    pin.on()
    motor.off()#to turn motor off
else:
    print("filling ")
    pin.off()
    motor.on()#to turn motor on

#-----
sleep(0.1)

```

#This code is to make the work automatically based on water level we can also integrate many sensors and for your reference I'm giving the wokwi link

#### NOTE:

Due to unavailability of libraries for micro python on Raspberry Pi Pico we cannot able to transfer the data from sensor to cloud platform .

Such as urequest, WIFI , think speak libraries

So, we used ESP32 & C/C++ programming language to work our Final project on real time data transfer.

<https://wokwi.com/projects/379535806439664641>-----  
-----

### Code in C/C++

```

#define BLYNK_TEMPLATE_ID "TMPL3-VBCPYBi"
#define BLYNK_TEMPLATE_NAME "water fountain"
#define BLYNK_AUTH_TOKEN "81q5QUnUB7tqWGrbsLJgcfrIsAqleTnF"

#define BLYNK_PRINT Serial

#include <WiFi.h>

```



```

#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>

char ssid[] = "Wokwi-GUEST";
char pass[] = "";

BlynkTimer timer;

// This function is called every time the Virtual Pin 0 state changes
BLYNK_WRITE(V0)
{
    // Set incoming value from pin V0 to a variable
    int value = param.asInt();

    // Update state
    Blynk.virtualWrite(V1, value);
}

// This function is called every time the device is connected to the
Blynk.Cloud
BLYNK_CONNECTED()
{
    // Change Web Link Button message to "Congratulations!"
    Blynk.setProperty(V3, "offImageUrl", "https://static-
image.nyc3.cdn.digitaloceanspaces.com/general/fte/congratulations.png");
    Blynk.setProperty(V3, "onImageUrl", "https://static-
image.nyc3.cdn.digitaloceanspaces.com/general/fte/congratulations_pressed.png"
);
    Blynk.setProperty(V3, "url", "https://docs.blynk.io/en/getting-started/what-
do-i-need-to-blynk/how-quickstart-device-was-made");
}

// This function sends Arduino's uptime every second to Virtual Pin 2.
void myTimerEvent()
{
    // You can send any value at any time.
    // Please don't send more that 10 values per second.
    Blynk.virtualWrite(V2, millis() / 1000);
}

#define Fountain_motor 17
#define Motor 4
#define triggerpin1 27
#define echopin1 26
float a;

```

```

int san();

void setup(){
  Serial.begin(9600);
  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);
  pinMode(trigerpin1, OUTPUT);
  pinMode(echopin1, INPUT);
  pinMode(Fountain_motor, OUTPUT);
  pinMode(Motor, OUTPUT);
  //used to measure duration of clock pulses

  Serial.println("hello sandy");
}

void loop(){
  a=san(trigerpin1,echopin1);
  Serial.print("distance (in cm)=");
  Serial.println(a);
  Blynk.virtualWrite(V0, a);
  delay(2000);

  //To show water level of the fountain
  Blynk.virtualWrite(V3, 4);
  if(a>10&&a<30){
    Blynk.virtualWrite(V3, 3);
  }else if(a>30&&a<50){
    Blynk.virtualWrite(V3, 2);
  }else if(a>50&&a<70){
    Blynk.virtualWrite(V3, 1);
  }else if(a>70){
    Blynk.virtualWrite(V3, 0);
  }

  //To turn on and of the fountain automatically
  if(a<10){
    digitalWrite(Fountain_motor, HIGH);
    digitalWrite(Motor, LOW);
    Blynk.virtualWrite(V1, 1); //foun
    Blynk.virtualWrite(V2, 0);
    Blynk.virtualWrite(V4, "Fountain Pump ON");
  }
  else{
    digitalWrite(Motor, HIGH);
    digitalWrite(Fountain_motor, LOW);
    Blynk.virtualWrite(V1, 0); //foun

```

```

    Blynk.virtualWrite(V2, 1);
    Blynk.virtualWrite(V4, "Filling Reservoir");
  }
}
int san(int triggerpin, int echopin){
  float distance, duration;
  digitalWrite(triggerpin, LOW);
  delay(2);
  digitalWrite(triggerpin, HIGH);
  delay(10);
  digitalWrite(triggerpin, LOW);
  duration=pulseIn(echopin,HIGH);
  distance=duration/2*0.034;
  //Serial.print("distance (in cm)=");
  //Serial.println(distance);
  delay(10);
  return distance;
}

```

## 10.Working Principle

Control and Monitor:

With the software in place, We can now control and monitor your smart water fountain from a smartphone or computer. we can turn the fountain on/off remotely, check water levels, and receive alerts if water levels drop too low or if there are temperature issues.

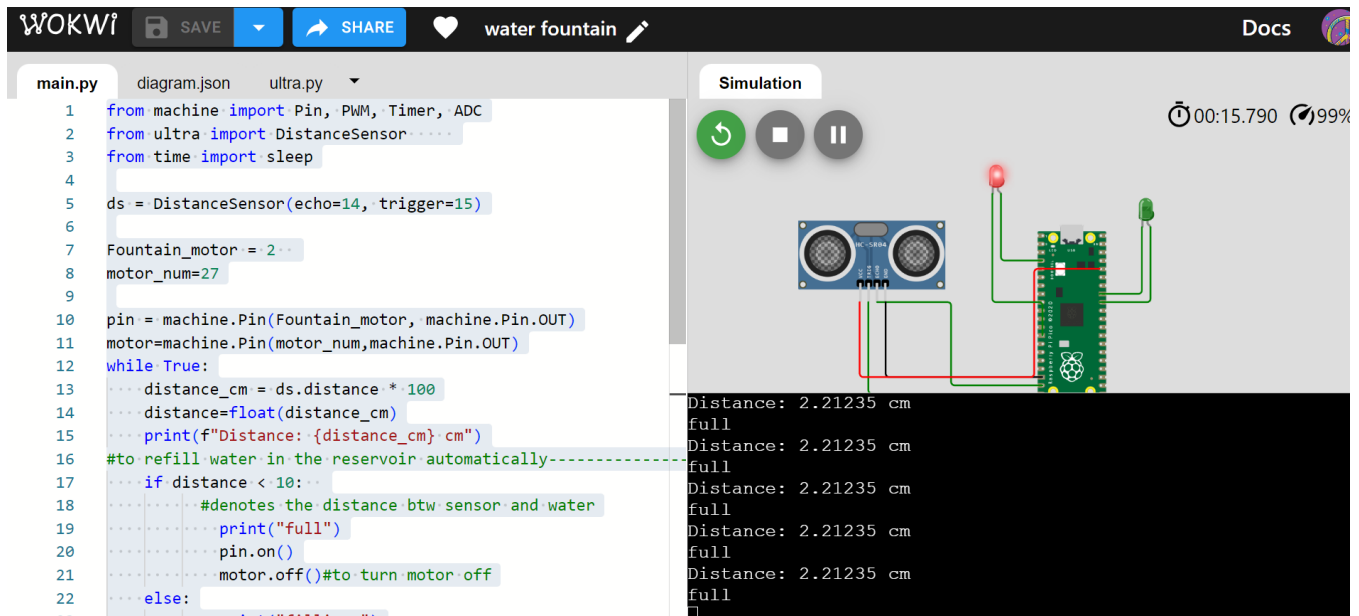
Automation:

We can automate our smart water fountain. For example, we could set schedules for the fountain to turn on and off automatically or create triggers based on environmental conditions.

Enclosure and Maintenance:

Ensure that all components are housed in a waterproof enclosure to protect them from water damage. Regularly check and maintain the fountain, including cleaning the pump and sensors.

## Simulation Using Raspberry pi



The screenshot shows the Wokwi web interface for a Raspberry Pi simulation. The top bar includes the Wokwi logo, a 'SAVE' button, a 'SHARE' button, a heart icon, the project name 'water fountain', and a 'Docs' link. The main area is divided into three panels: a code editor on the left, a simulation window in the middle, and a terminal on the right.

**Code Editor (main.py):**

```
1 from machine import Pin, PWM, Timer, ADC
2 from ultra import DistanceSensor
3 from time import sleep
4
5 ds = DistanceSensor(echo=14, trigger=15)
6
7 Fountain_motor = 2
8 motor_num=27
9
10 pin = machine.Pin(Fountain_motor, machine.Pin.OUT)
11 motor=machine.Pin(motor_num,machine.Pin.OUT)
12 while True:
13     distance_cm = ds.distance * 100
14     distance=float(distance_cm)
15     print(f"Distance: {distance_cm} cm")
16     #to refill water in the reservoir automatically-----
17     if distance < 10:
18         #denotes the distance btw sensor and water
19         print("full")
20         pin.on()
21         motor.off()#to turn motor off
22     else:
```

**Simulation Window:** Shows a visual representation of the hardware. A blue ultrasonic sensor module is connected to a Raspberry Pi. A red LED is connected to the Pi's GPIO pins, and a green LED is also connected. The sensor is emitting a red beam towards the Pi.

**Terminal:** Displays the output of the code, showing the distance measured by the sensor and the state of the motor and LEDs.

```
Distance: 2.21235 cm
full
Distance: 2.21235 cm
full
Distance: 2.21235 cm
full
Distance: 2.21235 cm
full
Distance: 2.21235 cm
full
Distance: 2.21235 cm
full
```

The red LED show the Fountain water pump running, So when reservoir is full the fountain will work

The Green LED shows the source pump to reservoir when the distance between water and the sensor increases the pump will start and fill the reservoir automatically

## Using C/C++ & BLYNK

WOKWI

SAVE

SHARE

new waterfountain ibm

Docs

sketch.ino

diagram.json

libraries.txt

Library Manager

```

1  #define BLYNK_TEMPLATE_ID "TMPL3-V8CPYBi"
2  #define BLYNK_TEMPLATE_NAME "water fountain"
3  #define BLYNK_AUTH_TOKEN "81q5QUnUB7tqWGrbsLJgcfRIsAqleTnF"
4
5  #define BLYNK_PRINT Serial
6  #include <WiFi.h>
7  #include <WiFiClient.h>
8  #include <BlynkSimpleEsp32.h>
9
10 char ssid[] = "Wokwi-GUEST";
11 char pass[] = "";
12
13 BlynkTimer timer;
14
15 // This function is called every time the Virtual Pin 0 state
16 BLYNK_WRITE(V0)
17 {
18   // Set incoming value from pin V0 to a variable
19   int value = param.asInt();
20
21   // Update state
22   Blynk.virtualWrite(V1, value);
23 }

```

Simulation

00:52.955

67%

distance (in cm)=1.00  
distance (in cm)=1.00  
distance (in cm)=1.00  
distance (in cm)=1.00  
distance (in cm)=1.00  
distance (in cm)=1.00  
distance (in cm)=1.00

water fountain

Online

Santhosh

My organization - 3699MO

Add Tag

Dashboard

Timeline

Device Info

Metadata

Actions Log

Latest

Last Hour

6 Hours

1 Day

1 Week

1 Month

3 Months

6 Months

1 Year

Custom

Yet to fill(in Cm)

2

0

200

Fountain

1

FOUNTAIN STATUS

Fountain Pump ON

Refill motor

0

WATER LEVEL 0-4

4

0

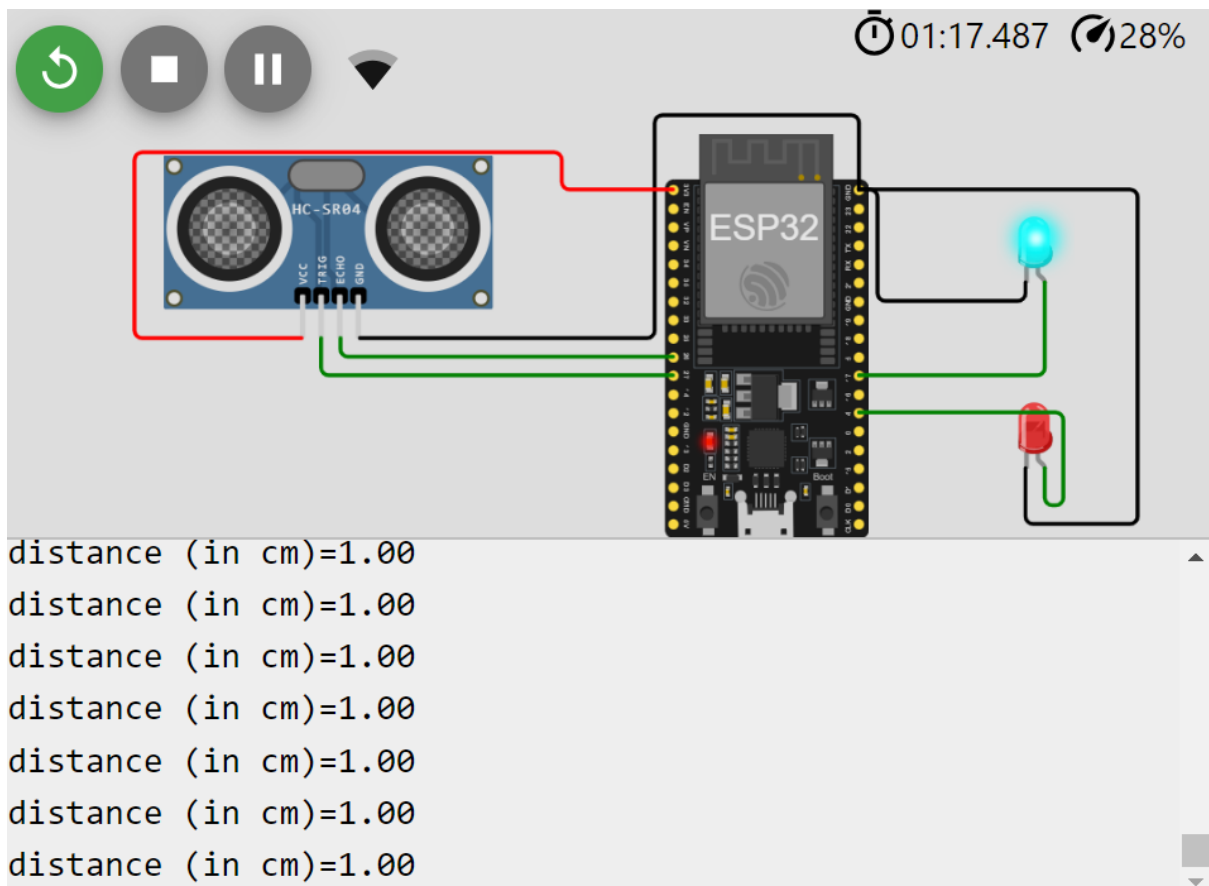
4

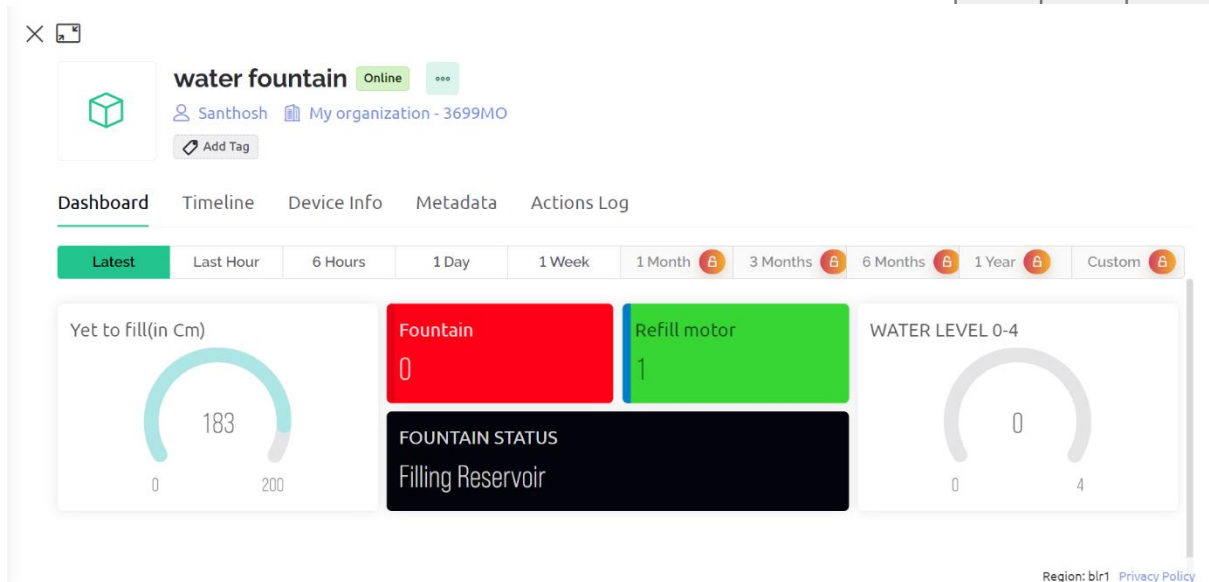
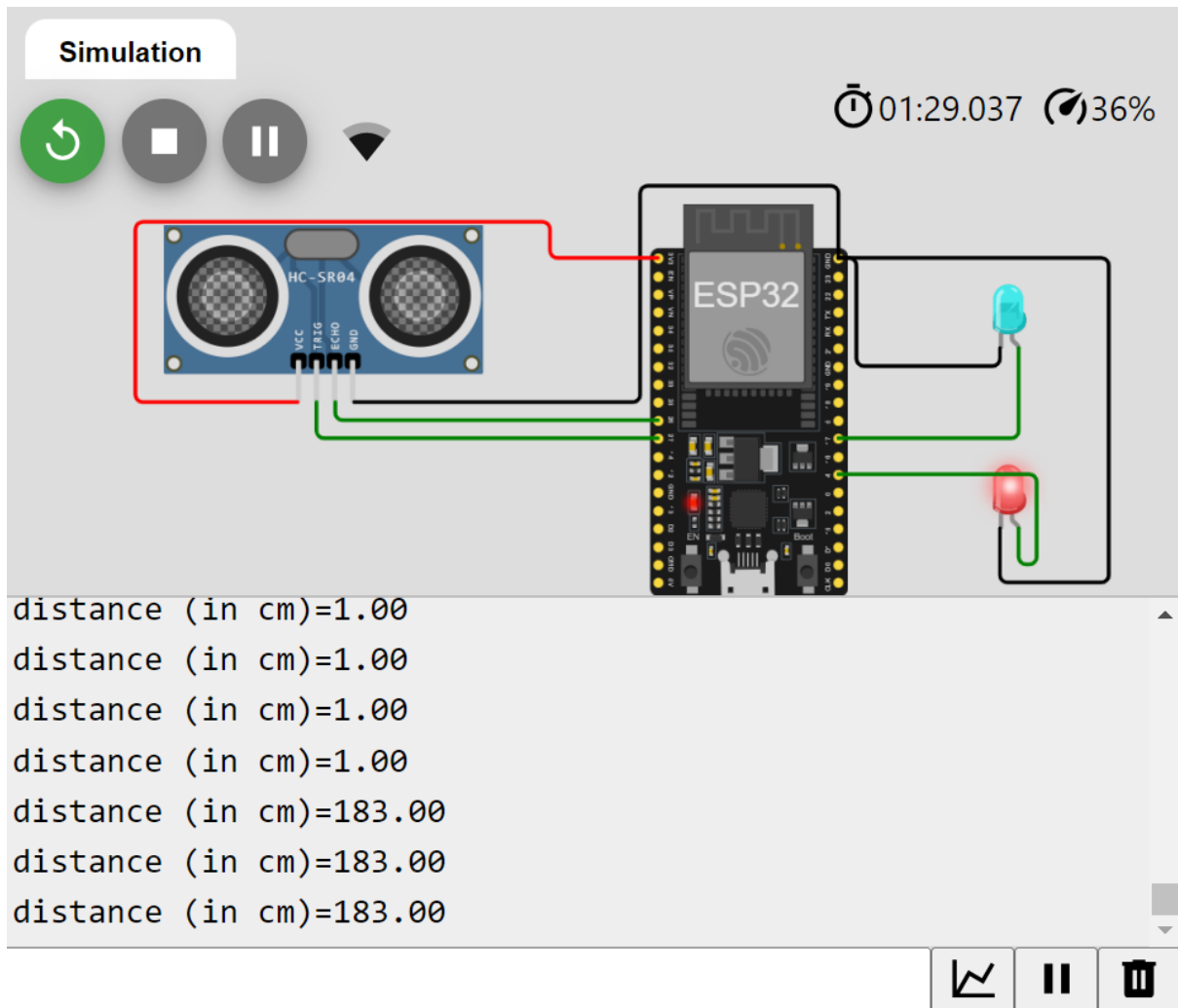
Region: blr1 Privacy Policy

The Blue LED indicated the functioning of the Fountain and the Red LED indicates the functioning of refill motor which fills the reservoir.

## BLYNK

We have used Blynk app to get live data from the Smart water Fountain using Api.





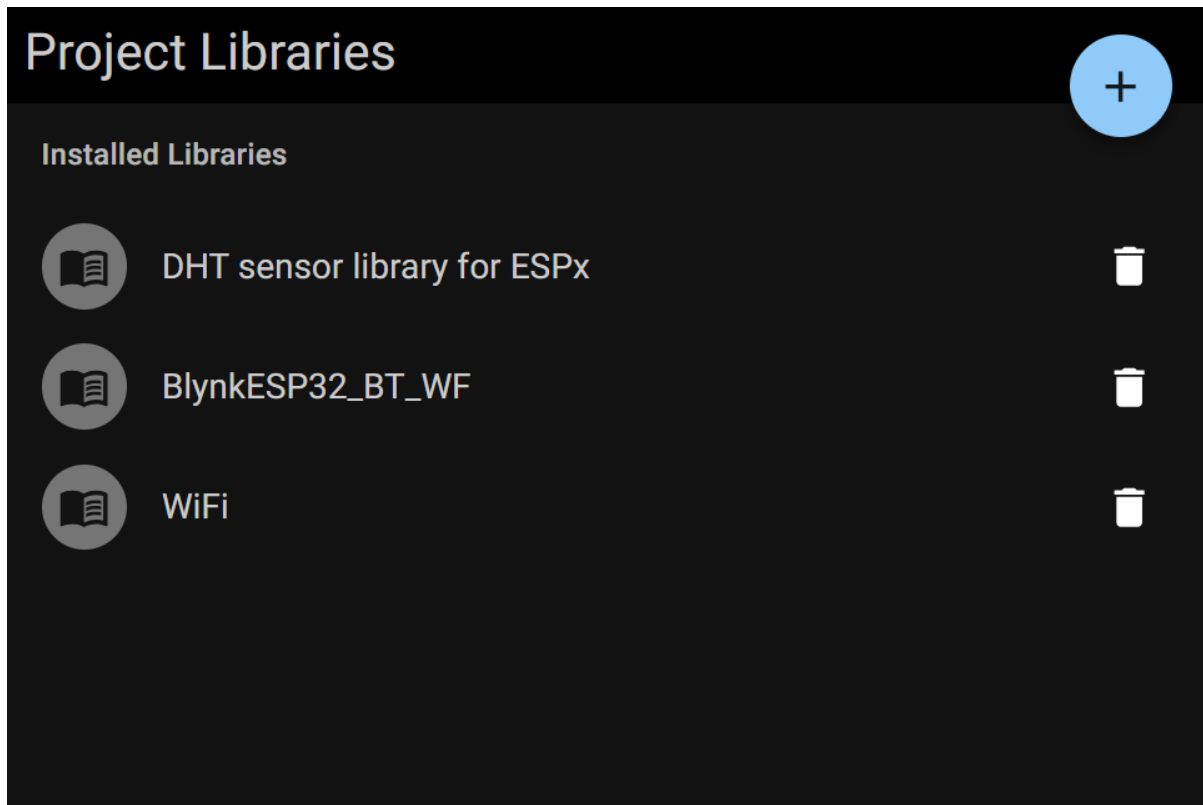
The Above processes can be monitored and controlled using BLYNK

## 11.Development Steps:

Step-1: Selecting the Components.

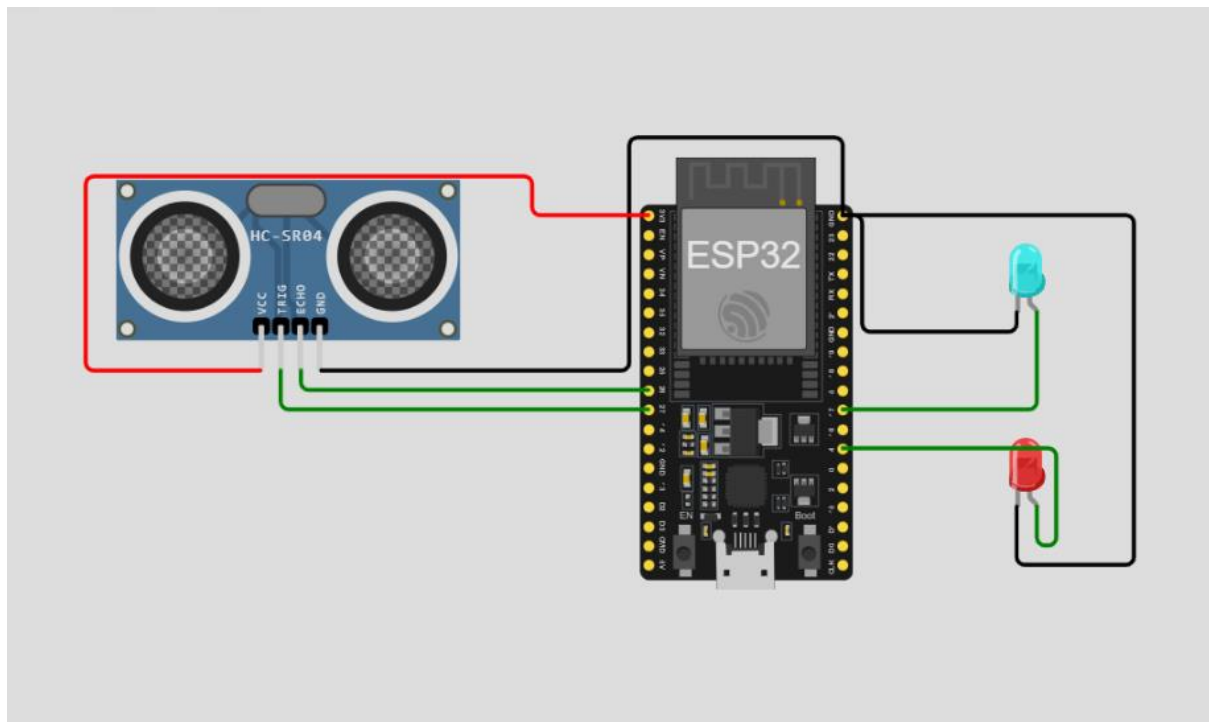
We Chose ESP32 & Ultrasonic(HC-SR04) sensor.

Step-2: Selecting the Required libraries.



Step-3: Connecting Components.





#### Step-4: Developing C/C++ Script.

```
//You can simulate this code using wokwi
//diagram.json &libraries.txt attached in same folder
//Happy Learning
//LINK---https://wokwi.com/projects/379912270128056321
//For more IOT projects
//https://github.com/santh0sh05/IoT\_projects.git

//replace with your Blynk credentials
#define BLYNK_TEMPLATE_ID "TMPL3-VBCPYBi"
#define BLYNK_TEMPLATE_NAME "water fountain"
#define BLYNK_AUTH_TOKEN "81q5QUnUB7tqWGrbsLJgcfrIsAqleTnF"

#define BLYNK_PRINT Serial
#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>

//Your wifi credentials
char ssid[] = "Wokwi-GUEST";
char pass[] = "";

BlynkTimer timer;
```

Step-5: Connecting to wifi & Blynk.

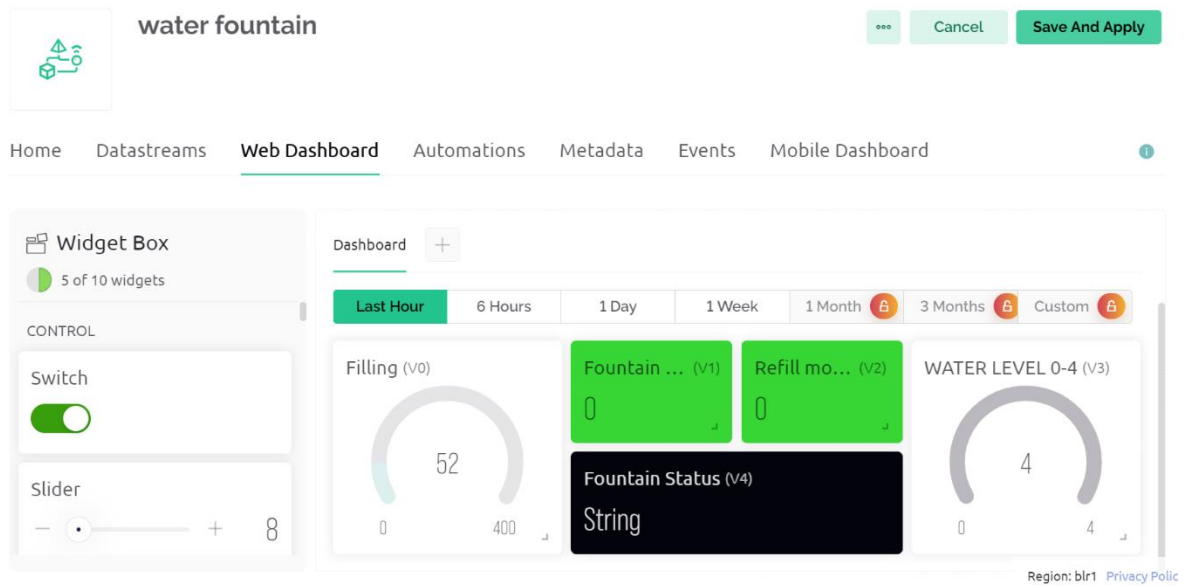
```
mode:DIO, clock div:2
load:0x3fff0030,len:1156
load:0x40078000,len:11456
ho 0 tail 12 room 4
load:0x40080400,len:2972
entry 0x400805dc
[1179] Connecting to Wokwi-GUEST

  _ _ _
 / _ )/ / _ _ _ _ / / _
 / _ / / / / _ \ / ' /
/ _ _ / _ \ , / / / _ \ \
    / _ _ / v1.3.2 on ESP32

#StandWithUkraine    https://bit.ly/swua

[3548] Connecting to blynk.cloud:80
[4435] Redirecting to blr1.blynk.cloud:80
[4440] Connecting to blr1.blynk.cloud:80
[5151] Ready (ping: 113ms).
```

Step-6: Designing Blynk Dashboard.



## 12. Advantages

- Smart fountains can monitor water quality in real-time, ensuring that the water is safe and clean for consumption.
- Features such as adjustable flow rates and automatic shut-off help in conserving water, especially in public spaces.
- Smart fountains can collect valuable data on water usage and quality, helping with maintenance and water resource management.
- Maintenance personnel can remotely monitor and control the fountains, which can reduce the cost and effort required for upkeep.

## 13. Conclusion

In this document, we have outlined an Internet of Things (IoT) project focused on optimizing water usage in smart water fountains to promote water conservation and responsible resource usage. By following a design thinking approach, water management authorities and environmentalists can leverage automation, ultimately leading to efficient water usage, conservation, and a more sustainable future.