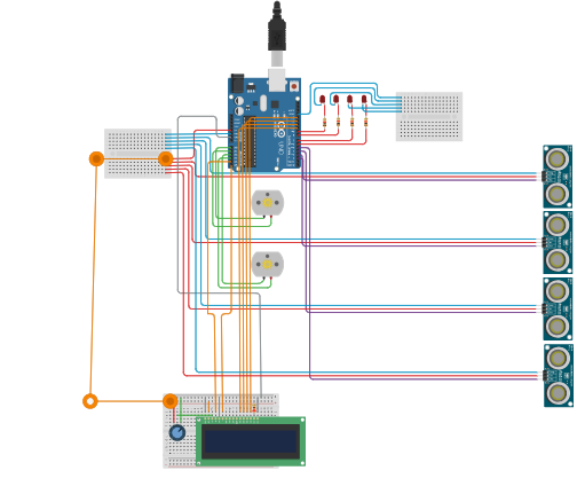
**TOPIC : SMART PUBLIC RESTROOM**

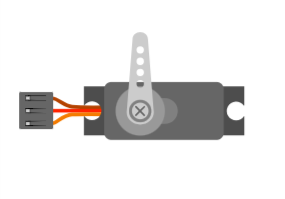
**CIRCUIT DIAGRAM**

**+**

**ARDUINO-UNO REFERENCES**

**servo Reference**

A standard Micro Servo Motor



## **Pin names**[**​**](https://docs.wokwi.com/parts/wokwi-servo#pin-names)

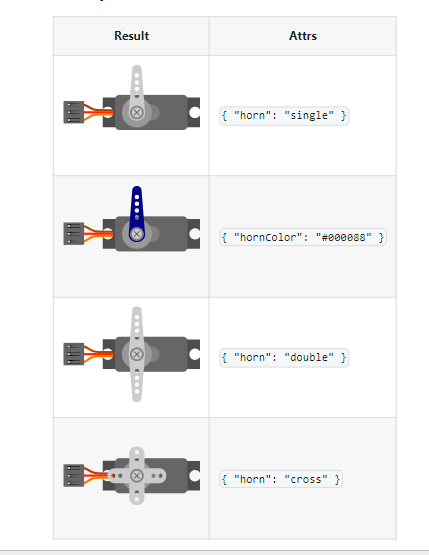
| **Name** | **Description** |
| --- | --- |
| PWM | Servo control signal |
| V+ | Positive voltage (5V) |
|  |  |
| GND | Ground |

## 

## **Attributes**[**​**](https://docs.wokwi.com/parts/wokwi-servo#attributes)

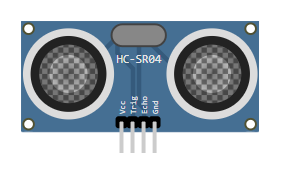
| **Name** | **Description** | **Default value** |
| --- | --- | --- |
| horn | Horn type: "single", "double" or "cross" | "single" |
| hornColor | The color of the servo's horn | "#ccc" |

**Example**

****

# hc-sr04 Reference

HC-SR04 Ultrasonic Distance Sensor

****

## **Pin names**[**​**](https://docs.wokwi.com/parts/wokwi-hc-sr04#pin-names)

| **Name** | **Description** |
| --- | --- |
| VCC | Voltage supply (5V) |
| TRIG | Pulse to start the measurement |
| ECHO | Measure the high pulse length to get the distance |
| GND | Ground |

## **Attributes**

## [**​**](https://docs.wokwi.com/parts/wokwi-hc-sr04#attributes)

| **Name** | **Description** | **Default value** |
| --- | --- | --- |
| distance | Initial distance value, in centimeters | "400" |

## **Operation**[**​**](https://docs.wokwi.com/parts/wokwi-hc-sr04#operation)

To start a new distance measurement set the TRIG pin to high for 10uS or more. Then wait until the ECHO pin goes high, and count the time it stays high (pulse length). The length of the ECHO high pulse is proportional to the distance. Use the following table to convert the ECHO pulse length in microseconds into centimeters / inches

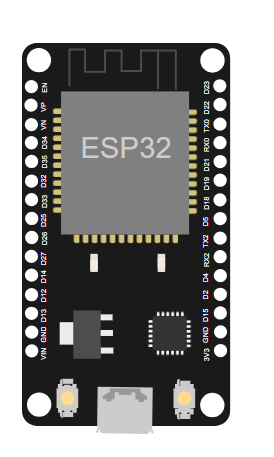
| **Unit** | **Distance** |
| --- | --- |
| Centimeters | PulseMicros / 58 |
| Inches | PulseMicros / 148 |

### **Setting the distance**[**​**](https://docs.wokwi.com/parts/wokwi-hc-sr04#setting-the-distance)

To change the distance while the simulation is running, click on the HC-SR04 drawing in the diagram and use the slider to set the distance value. You can choose any value between 2cm and 400cm.

# ESP32 Simulation

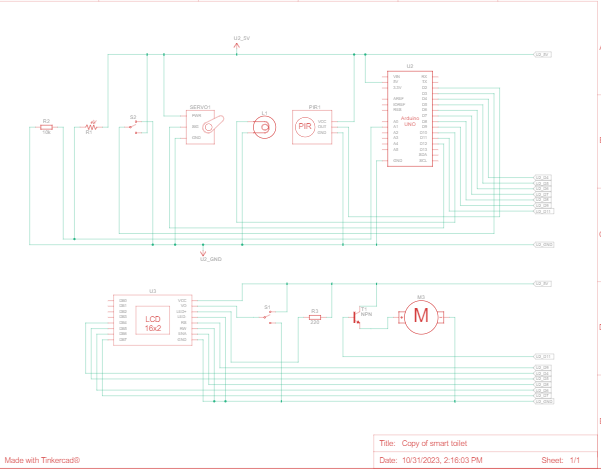
# The ESP32 is a popular WiFi and Bluetooth-enabled microcontroller, widely used for IoT Projects. Wokwi simulates the ESP32, ESP32-C3, ESP32-S2, ESP32-S3, ESP32-C6 (beta), and ESP32-H2 (alpha).

****

| **Name** | **Chip** | **Description** |
| --- | --- | --- |
| ESP32 DevKit v1 | ESP32 | Popular ESP32 development board |
| ESP32-S2-DevKitM-1 | ESP32-S2 | Entry-level ESP32-S2 development board |
| [Franzininho WiFi](https://docs.wokwi.com/parts/board-franzininho-wifi) | ESP32-S2 | Board by the Franzininho Community |
| [Wemos S2 mini](https://wokwi.com/projects/355047217294313473) | ESP32-S2 | Small ESP32-S2 board by Wemos |
| ESP32-S3-DevKitC-1 | ESP32-S3 | Entry-level ESP32-S3 development board |
| ESP32-C3-DevKitM-1 | ESP32-C3 | Entry-level ESP32-C3 development board |
| Rust Board ESP32-C3 | ESP32-C3 | ESP32-C3 board designed for [Rust trainings](https://github.com/esp-rs/std-training) |
| ESP32-C6-DevKitC-1 | ESP32-C6 | Entry-level ESP32-C6 development board (beta) |
| ESP32-H2-DevKitM-1 | ESP32-H2 | Entry-level ESP32-H2 development board (alpha) |

You can contribute additional boards by sending a pull request to [wokwi-boards](https://github.com/wokwi/wokwi-boards).

**SCHEMATIC DIAGRAM :**

****

**RASPBERRY PI CODE :**

import RPi.GPIO as GPIO

import time

import requests

# Configure GPIO pins

GPIO.setmode(GPIO.BCM)

OCCUPANCY\_SENSOR\_PIN = 4

CLEANING\_ALERT\_PIN = 17

LED\_PIN = 18

GPIO.setup(OCCUPANCY\_SENSOR\_PIN, GPIO.IN)

GPIO.setup(CLEANING\_ALERT\_PIN, GPIO.IN)

GPIO.setup(LED\_PIN, GPIO.OUT)

# REST API endpoint for sending cleaning alerts

CLEANING\_ALERT\_API = "http://your\_server/cleaning-alert"

# Function to check occupancy and trigger LED

def check\_occupancy(channel):

if GPIO.input(OCCUPANCY\_SENSOR\_PIN):

print("Restroom is occupied.")

GPIO.output(LED\_PIN, GPIO.HIGH)

else:

print("Restroom is vacant.")

GPIO.output(LED\_PIN, GPIO.LOW)

# Function to send cleaning alerts

def send\_cleaning\_alert(channel):

if GPIO.input(CLEANING\_ALERT\_PIN):

print("Cleaning alert sent.")

requests.post(CLEANING\_ALERT\_API, data={"alert": "Cleaning needed"})

# Add event detection

GPIO.add\_event\_detect(OCCUPANCY\_SENSOR\_PIN, GPIO.BOTH, callback=check\_occupancy, bouncetime=300)

GPIO.add\_event\_detect(CLEANING\_ALERT\_PIN, GPIO.BOTH, callback=send\_cleaning\_alert, bouncetime=300)

try:

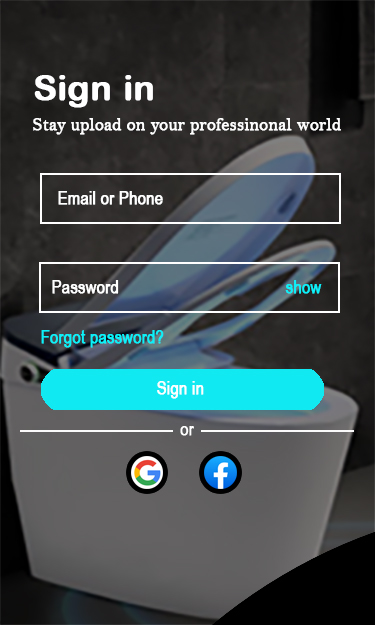
while True:

time.sleep(1)

except KeyboardInterrupt:

GPIO.cleanup()

**Mobile App**

****

****