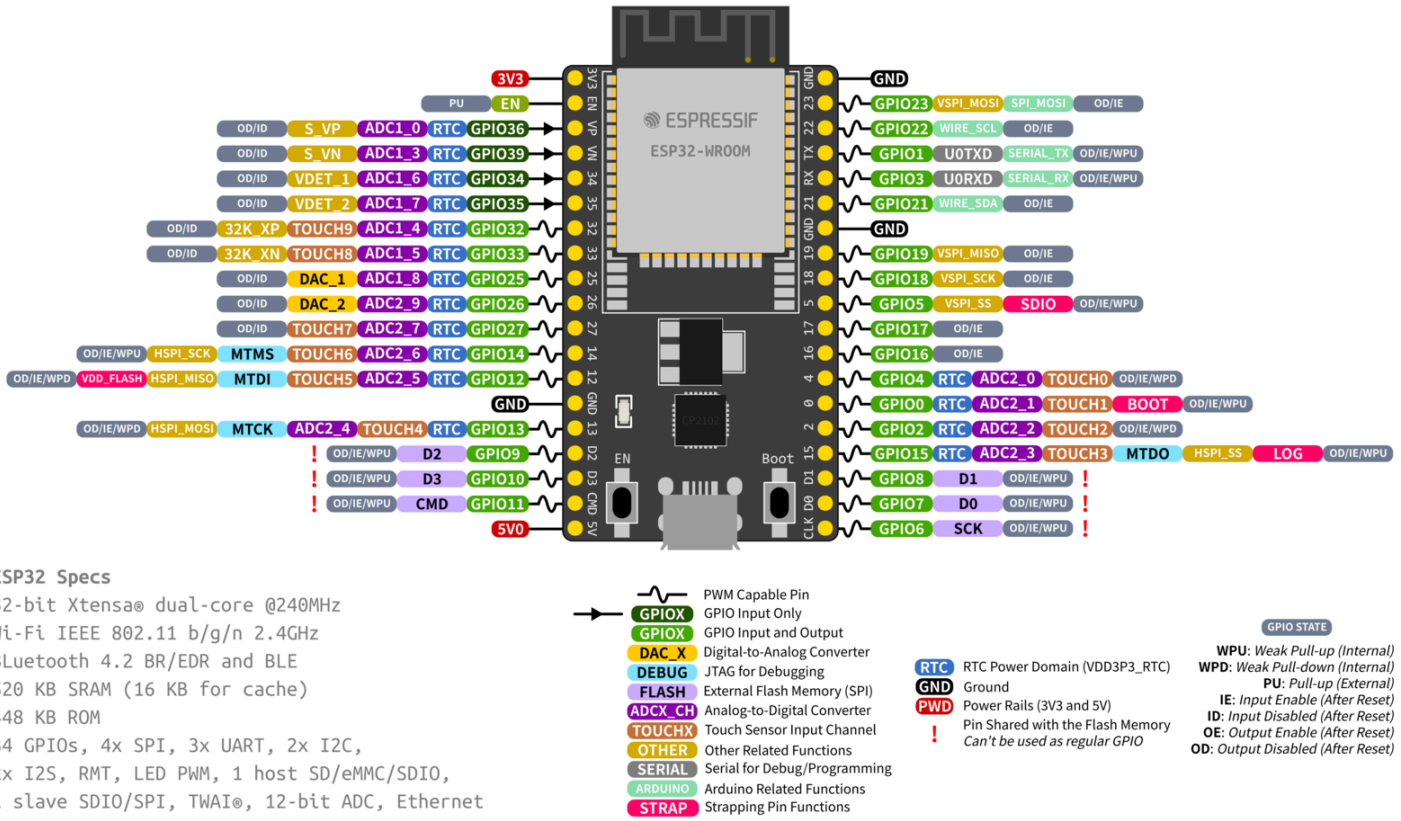
BOARD: DOIT ESP32 DEVKIT V1
FRAMEWORK: ARDUINO



PLATFORM.INI

```
; PlatformIO Project Configuration File
;
;   Build options: build flags, source filter
;   Upload options: custom upload port, speed and extra flags
;   Library options: dependencies, extra library storages
;   Advanced options: extra scripting
;
; Please visit documentation for the other options and examples
; https://docs.platformio.org/page/projectconf.html

[env:esp32doit-devkit-v1]
platform = espressif32
board = esp32doit-devkit-v1
framework = arduino
monitor_speed = 115200
lib_deps =
    me-no-dev/ESPAsyncTCP@^1.2.2
    https://github.com/adafruit/DHT-sensor-library/archive/refs/heads/master.zip
    madhephaestus/ESP32Servo@^1.1.2
    me-no-dev/ESP Async WebServer@^1.2.4
    me-no-dev/AsyncTCP@^1.1.1
```

CPP CODE

```cpp
#include <Arduino.h>
#include "WiFi.h"
#include "ESPAsyncWebServer.h"
#include "SPIFFS.h"
#include <DHT.h>
#include <Adafruit_Sensor.h>
#include <ESP32Servo.h>

#define DHTPIN 22      // Digital pin connected to the DHT sensor
#define DHTTYPE DHT22  // DHT 22
#define SOIL_MOISTURE_PIN 34 // Analog pin connected to the soil moisture sensor
#define TRIGGER_PIN 5  // Pin connected to HC-SR04 trigger
#define ECHO_PIN 18    // Pin connected to HC-SR04 echo
/*#define RGB_LED_PIN_R 25 // Pin connected to RGB LED - Red
#define RGB_LED_PIN_G 26 // Pin connected to RGB LED - Green
#define RGB_LED_PIN_B 27 // Pin connected to RGB LED - Blue*/

// Replace with your network credentials
const char *ssid = "সরদার";
const char *password = "1111rasel";

const int ledPin2 = 19;       // Pin for LED
const int ledPin1 = 21;       // New LED pin
const int photoresistorPin = 36; // Analog pin connected to the photoresistor
const int WATER_LEVEL_PIN = 39;  // Analog pin connected to the water level sensor
const int relayPin1 = 23;     // Relay pin for light intensity
const int relayPin2 = 32;     // Relay pin for water level
const int servoPin = 13;      // Pin connected to the servo motor

int lightIntensity = 0;
int waterLevel = 0;
int soilMoisture = 0;
float temperature = 0.0;
float humidity = 0.0;
int distance = 0;
Servo myservo;
String ledState2;
String ledState1; // State of ledPin
DHT dht(DHTPIN, DHTTYPE);
Servo servo;
int angle;

AsyncWebServer server(80);
```

```
String processor(const String &var) {
  Serial.println(var);
  if (var == "LIGHT_INTENSITY") {
    return String(lightIntensity);
  } else if (var == "WATER_LEVEL") {
    return String(waterLevel);
  } else if (var == "TEMPERATURE") {
    return String(temperature);
  } else if (var == "HUMIDITY") {
    return String(humidity);
  } else if (var == "SOIL_MOISTURE") {
    return String(soilMoisture);
  } else if (var == "DISTANCE") {
    return String(distance);
 } else if (var == "SERVO_ANGLE") {
    return String(myservo.read());
  } else if (var == "STATE2") {
    if (digitalRead(ledPin2)) {
      ledState2 = "ON";
    } else {
      ledState2 = "OFF";
    }
    Serial.println(ledState2);
    return ledState2;
  } else if (var == "STATE1") { // New block to handle ledPin2
    if (digitalRead(ledPin1)) {
      ledState1 = "ON";
    } else {
      ledState1 = "OFF";
    }
    Serial.println(ledState1);
    return ledState1;
  }
  return String();
}

void setup() {
  // Serial port for debugging purposes
  Serial.begin(115200);
  pinMode(ledPin2, OUTPUT);
  pinMode(ledPin1, OUTPUT);
  pinMode(photoresistorPin, INPUT);
  pinMode(relayPin1, OUTPUT);
  pinMode(relayPin2, OUTPUT);
  /*pinMode(RGB_LED_PIN_R, OUTPUT);
  pinMode(RGB_LED_PIN_G, OUTPUT);
  pinMode(RGB_LED_PIN_B, OUTPUT);*/
  pinMode(SOIL_MOISTURE_PIN, INPUT);
```

```
pinMode(TRIGGER_PIN, OUTPUT);
pinMode(ECHO_PIN, INPUT);
servo.attach(servoPin);

// Initialize SPIFFS
if (!SPIFFS.begin(true)) {
  Serial.println("An Error has occurred while mounting SPIFFS");
  return;
}

WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.println("Connecting to WiFi..");
}

Serial.println(WiFi.localIP());

server.on("/", HTTP_GET, [](AsyncWebServerRequest *request) {
  request->send(SPIFFS, "/index.html", String(), false, processor);
});

server.on("/style.css", HTTP_GET, [](AsyncWebServerRequest *request) {
  request->send(SPIFFS, "/style.css", "text/css");
});

server.on("/on2", HTTP_GET, [](AsyncWebServerRequest *request) {
  digitalWrite(ledPin2, HIGH);
  request->send(SPIFFS, "/index.html", String(), false, processor);
});

server.on("/off2", HTTP_GET, [](AsyncWebServerRequest *request) {
  digitalWrite(ledPin2, LOW);
  request->send(SPIFFS, "/index.html", String(), false, processor);
});
  server.on("/on1", HTTP_GET, [](AsyncWebServerRequest *request) {
  digitalWrite(ledPin1, HIGH);
  request->send(SPIFFS, "/index.html", String(), false, processor);
});

server.on("/off1", HTTP_GET, [](AsyncWebServerRequest *request) {
  digitalWrite(ledPin1, LOW);
  request->send(SPIFFS, "/index.html", String(), false, processor);
});

// Route to retrieve light intensity
server.on("/light_intensity", HTTP_GET, [](AsyncWebServerRequest *request) {
  request->send_P(200, "text/plain", String(lightIntensity).c_str());
```

```
  });

  // Route to retrieve water level
  server.on("/water_level", HTTP_GET, [](AsyncWebServerRequest *request) {
    request->send_P(200, "text/plain", String(waterLevel).c_str());
  });

  // Route to retrieve temperature
  server.on("/temperature", HTTP_GET, [](AsyncWebServerRequest *request) {
    request->send_P(200, "text/plain", String(temperature).c_str());
  });

  // Route to retrieve humidity
  server.on("/humidity", HTTP_GET, [](AsyncWebServerRequest *request) {
    request->send_P(200, "text/plain", String(humidity).c_str());
  });

  // Route to retrieve soil moisture
  server.on("/soil_moisture", HTTP_GET, [](AsyncWebServerRequest *request) {
    request->send_P(200, "text/plain", String(soilMoisture).c_str());
  });

  // Route to retrieve ultrasonic distance
  server.on("/distance", HTTP_GET, [](AsyncWebServerRequest *request) {
    request->send_P(200, "text/plain", String(distance).c_str());
  });

  // Route to retrieve servo angle
  server.on("/servo_angle", HTTP_GET, [](AsyncWebServerRequest *request) {
    request->send_P(200, "text/plain", String(servo.read()).c_str());
  });

  server.begin();

  // Initialize DHT sensor
  Serial.println("DHT Initializing...");
  dht.begin();
}

void loop() {

  int sensorValue = analogRead(photoresistorPin);
  lightIntensity = map(sensorValue, 0, 4095, 0, 100);
  delay(100); // wait 10 milliseconds
  waterLevel = analogRead(WATER_LEVEL_PIN); // Read the analog value from sensor
  delay(100); // wait 10 milliseconds
  soilMoisture = analogRead(SOIL_MOISTURE_PIN); // Read the soil moisture sensor
value
```

```
  delay(100); // wait 10 milliseconds

  // Read ultrasonic distance
  digitalWrite(TRIGGER_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIGGER_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIGGER_PIN, LOW);
  distance = pulseIn(ECHO_PIN, HIGH) * 0.034 / 2;

 // Read DHT sensor
  humidity = dht.readHumidity();
  temperature = dht.readTemperature();

  if (isnan(humidity) || isnan(temperature))
  {
    Serial.println("Failed to read from DHT sensor!");
  }
  else
  {
    Serial.print("Humidity: ");
    Serial.print(humidity);
    Serial.print(" %\t");
    Serial.print("Temperature: ");
    Serial.print(temperature);
    Serial.println(" *C");
  }

  // Control servo based on ultrasonic distance
if (distance >= 0 && distance <= 30)
  {
    // If distance is between 0 and 30 cm, rotate servo to 90 degrees
    servo.write(90);
  }
  else
  {
    // Otherwise, keep servo at 0 degrees
    servo.write(0);
  }
  // Update web server with current servo angle
  server.on("/servo_angle", HTTP_GET, [](AsyncWebServerRequest *request) {
    request->send_P(200, "text/plain", String(servo.read()).c_str());
  });

  // Control relay based on light intensity
  if (lightIntensity <= 50 || digitalRead(ledPin1) == HIGH)
  {
    digitalWrite(relayPin1, HIGH); // Turn on relay
```

```
  }
  else if (lightIntensity > 50 || digitalRead(ledPin1) == LOW)
  {
    digitalWrite(relayPin1, LOW); // Turn off relay
  }

  // Control relay based on water level
  if (waterLevel <= 30 || digitalRead(ledPin2) == HIGH)
  {
    digitalWrite(relayPin2, HIGH); // Turn on relay
  }
  else if (waterLevel >30 || digitalRead(ledPin2) == LOW)
  {
    digitalWrite(relayPin2, LOW); // Turn off pump relay
  }

  /*/* Change RGB LED color based on sensor values
  int redValue = map(temperature, 0, 50, 0,100); // Map temperature to red color
intensity
  int greenValue = map(humidity, 0, 100, 0, 100); // Map humidity to green color
intensity
  int blueValue = map(soilMoisture, 0, 4095, 100, 0); // Map soil moisture to blue
color intensity
  int red= max(redValue, max(greenValue, blueValue));
  int green= min(redValue, min(greenValue, blueValue));

  int blue = (red+green)/2;
   if(red <30 || green<30) {
  digitalWrite(RGB_LED_PIN_R, 1);   // Set red LED intensity
  digitalWrite(RGB_LED_PIN_G, 0); // Set green LED intensity
  digitalWrite(RGB_LED_PIN_B, 0);  // Set blue LED intensity
  }
 else if ( green>80 && red> 30 ) {
  digitalWrite(RGB_LED_PIN_R, 0);   // Set red LED intensity
  digitalWrite(RGB_LED_PIN_G, 0); // Set green LED intensity
  digitalWrite(RGB_LED_PIN_B, 1);  // Set blue LED intensity
 }
 else if (green<80 && red > 30 ) {
  digitalWrite(RGB_LED_PIN_R, 0);   // Set red LED intensity
  digitalWrite(RGB_LED_PIN_G, 1); // Set green LED intensity
  digitalWrite(RGB_LED_PIN_B, 0);  // Set blue LED intensity
 }*/
  delay(2000);
}
```

HTML CODE

```
<!DOCTYPE html>
<html>
```

```
<head>
    <title>ESP32 Web Server</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="icon" href="data:,">
    <link rel="stylesheet" type="text/css" href="style.css">
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css
" rel="stylesheet">
    <style>
        /* Your existing CSS styles */
        .dark-mode {
            background-color: #000000;
            color: #fff;
        }
    </style>
    <script>
     document.addEventListener("DOMContentLoaded", function() {
    const pinInput = document.getElementById("pinInput");
    const unlockBtn = document.getElementById("unlockBtn");
    const lockScreen = document.querySelector(".lock-screen");
    const homeScreen = document.querySelector(".home-screen");

    // Check if the lock screen should be displayed
    const isLocked = localStorage.getItem("isLocked");
    if (isLocked === null) {
        // Lock screen should be displayed for the first time
        lockScreen.style.display = "block";
        homeScreen.style.display = "none";
    } else {
        // Lock screen is not needed, show home screen directly
        homeScreen.style.display = "block";
        lockScreen.style.display = "none";

              updateLightIntensity();
              updateWaterLevel();
              updateTemperature();
              updateHumidity();
              updateSoilMoisture();
             updateDistance();
             updateServoAngle();
             updateRGB();

            setInterval(updateLightIntensity, 1000);
            setInterval(updateWaterLevel, 2300);
            setInterval(updateTemperature, 1400);
            setInterval(updateHumidity, 1600);
            setInterval(updateSoilMoisture, 1800);
            setInterval(updateDistance, 2000);
```

```
        setInterval(updateRGB, 2100);
}

unlockBtn.addEventListener("click", function() {
    const enteredPin = pinInput.value;
    const validPin = "1234"; // Change this to your desired PIN
    if (enteredPin === validPin) {
        // Unlock successful, set flag to indicate it's unlocked
        localStorage.setItem("isLocked", "false");

        lockScreen.style.opacity = "0";
        homeScreen.style.opacity = "1";
        setTimeout(() => {
            lockScreen.style.display = "none";
            homeScreen.style.display = "block";
            updateLightIntensity();
           updateWaterLevel();
           updateTemperature();
           updateHumidity();
           updateSoilMoisture();
          updateDistance();
          updateServoAngle();
          updateRGB();

        setInterval(updateLightIntensity, 1000);
        setInterval(updateWaterLevel, 2300);
        setInterval(updateTemperature, 1400);
        setInterval(updateHumidity, 1600);
        setInterval(updateSoilMoisture, 1800);
        setInterval(updateDistance, 2000);
        setInterval(updateRGB, 2100);
        }, 1000); // Adjust timing according to the transition duration
    } else {
        window.Text("Invalid PIN. Please try again.");
    pinInput.value = "";
    }
});

function updateLightIntensity() {
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        var lightIntensity = parseInt(this.responseText);
        document.getElementById("lightIntensity").innerHTML = lightIntensity;
        if (lightIntensity < 70) {
            document.body.classList.add("dark-mode");
        } else {
            document.body.classList.remove("dark-mode");
```

```
            }
        }
    };
    xhttp.open("GET", "/light_intensity", true);
    xhttp.send();
}

function updateWaterLevel() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            var waterLevel = parseInt(this.responseText);
            document.getElementById("waterLevel").innerHTML = waterLevel;

            const minWL = 0; // Minimum water level
            const maxWL = 4095; // Maximum water level
            const minHeight = 0; // Minimum height of water (bottom)
            const maxHeight = 200; // Maximum height of water (top)
            const height = minHeight + ((waterLevel - minWL) / (maxWL - minWL)) *
(maxHeight - minHeight);
            document.getElementById("water").style.height = height + "px";

            if (waterLevel < 30) {
                document.getElementById("relayStatus2").innerHTML = "ON";
            } else {
                document.getElementById("relayStatus2").innerHTML = "OFF";
            }
        }
    };
    xhttp.open("GET", "/water_level", true);
    xhttp.send();
}

function updateTemperature() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            var temperature = parseFloat(this.responseText);
            document.getElementById("temperature").innerHTML = temperature;

            const minTemp = 0; // Minimum temperature
            const maxTemp = 50; // Maximum temperature
            const minHeight = 0; // Minimum height of mercury (bottom)
            const maxHeight = 200; // Maximum height of mercury (top)
            const height = minHeight + ((temperature - minTemp) / (maxTemp -
minTemp)) * (maxHeight - minHeight);
            document.getElementById("mercury").style.height = height + "px";
        }
```

```
    };
    xhttp.open("GET", "/temperature", true);
    xhttp.send();
}

function updateHumidity() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            var humidity = parseFloat(this.responseText);
            document.getElementById("humidity").innerHTML = humidity;

            const minHum = 0; // Minimum humidity
            const maxHum = 100; // Maximum humidity
            const minWidth = 0; // Minimum width of scale (left)
            const maxWidth = 200; // Maximum width of scale (right)
            const width = minWidth + ((humidity - minHum) / (maxHum - minHum)) *
(maxWidth - minWidth);
            document.getElementById("drew").style.width = width + "px";
        }
    };
    xhttp.open("GET", "/humidity", true);
    xhttp.send();
}

function updateSoilMoisture() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            var soilMoisture = parseInt(this.responseText);
            document.getElementById("soilMoisture").innerHTML = soilMoisture;

            const minMos = 0; // Minimum soil moisture
            const maxMos = 4095; // Maximum soil moisture
            const minWidth = 0; // Minimum width of scale (left)
            const maxWidth = 200; // Maximum width of scale (right)
            const width = minWidth + ((soilMoisture - minMos) / (maxMos - minMos)) *
(maxWidth - minWidth);
            document.getElementById("soil").style.width = width + "px";
        }
    };
    xhttp.open("GET", "/soil_moisture", true);
    xhttp.send();
}

function updateDistance() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
```

```
        if (this.readyState == 4 && this.status == 200) {
            var distance = parseInt(this.responseText);
            document.getElementById("distance").innerHTML = distance;

            const minDis = 0; // Minimum distance
            const maxDis = 1201; // Maximum distance
            const minWidth = 0; // Minimum width of scale (left)
            const maxWidth = 200; // Maximum width of scale (right)
            const width = minWidth + ((distance - minDis) / (maxDis - minDis)) *
(maxWidth - minWidth);
            document.getElementById("tick").style.width = width + "px";
        }
    };
    xhttp.open("GET", "/distance", true);
    xhttp.send();
}

function updateServoAngle() {
    var xhr = new XMLHttpRequest();
    xhr.onreadystatechange = function() {
        if (xhr.readyState == 4 && xhr.status == 200) {
            var servoAngle = parseInt(xhr.responseText);
            document.getElementById("servoAngle").innerHTML = servoAngle;
            updateDoor(servoAngle);
        }
    };
    xhr.open("GET", "/servo_angle", true);
    xhr.send();
}

function updateDoor(servoAngle) {
    var panel = document.querySelector('.panel');
    panel.classList.toggle('open', servoAngle === 89);
}

setInterval(updateServoAngle, 1000);

function updateRGB() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            var rgb = parseInt(this.responseText);
            document.getElementById("rgb").innerHTML = rgb;
        }
    };
    xhttp.open("GET", "/rgb_led", true);
    xhttp.send();
}
```

```
  });
    </script>


</head>
<body>
    <div class="credit">TANVIR SHEAK, 5B</div>
    <div class="lock-screen">
        <img
src="https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEhFKVAvSKsjVGGi7vFBrZTRXUFjXcLdWhouI880_8DBha-IhNocVCHhIX2t_itLStEs6cxJdQI5Pn-91p_DR9bqvPvH23vzREjGnR_ReVYZjIm42FkCOy3R3xezP70JEQIzKUfGyi7pLvYWXZxOw6aXHTo79DqoWcnAnO_DcV3EYzaSbZwFBMOqOWXOdXlP/s1600/g_house.png" alt="Lock Screen Wallpaper" class="wallpaper">
        <div class="lock-content">
            <h1>WELCOME TO<br>OUR GREEN OASIS</h1>
        <h4>Enter PIN</h4>
          <input type="password" id="pinInput" maxlength="4">
          <button id="unlockBtn">Go</button>
        </div>
      </div>
    <div class="container home-screen" >
        <h2 class="mt-4">SMART PLANT <br>MONITORING AND CARE WITH ESP32
        </h2>
        <div class="row mt-4">
            <div class="col-md-4">
                <p>Air Humidity: <strong id="humidity">%HUMIDITY%</strong></p>
                <div class="scale" style="margin: 0 auto;">
      <div class="drew" id="drew"></div>

              </div>
              </div>
              <div class="col-md-4">
                <p>Soil Moisture: <strong
id="soilMoisture">%SOIL_MOISTURE%</strong></p>
                <div class="bench" style="margin: 0 auto;">
                  <div class="soil" id="soil"></div>

              </div>
              </div>
              <div class="col-md-4">
                <p>Door Proximity: <strong id="distance">%DISTANCE%</strong></p>
                <div class="line" style="margin: 0 auto;">
                  <div class="tick" id="tick"></div>

              </div>
              </div>
        </div>
        <div class="row mt-4">
```

```
            <!--div class="col-md-3">
                <p>GPIO state: <strong>%STATE%</strong></p>
            </div-->


            <div class="col-md-4">
                <p>Water Level: <strong id="waterLevel">%WATER_LEVEL%</strong></p>
                <div class="glass"  style="margin: 0 auto;">
                    <div class="water" id="water"></div>


                </div>
            </div>
            <div class="col-md-4">
                <p>Temperature: <strong id="temperature">%TEMPERATURE%</strong></p>
                <div class="thermometer" style="margin: 0 auto;">
                    <div class="mercury" id="mercury"></div>


                </div>
            </div>
            <div class="col-md-4">
                <p>Servo Angle: <strong id="servoAngle">%SERVO_ANGLE%</strong></p>
                <div class="door" style="margin: 0 auto;">
                    <div class="panel"></div>
                </div>
            </div>

        </div>

        <div class="row mt-4">
           <!-----<div class="col-md-3">
                <p>RGB value: <strong id="rgb">%RGB_LED%</strong></p>
            </div>  -->
            <div class="col-md-4">
                <p style="color: hsl(0, 100%, 50%); text-shadow: 8px 8px 10px rgb(0,
146, 61); -webkit-text-stroke: 1px black;">Light Intensity: <strong
id="lightIntensity" style="color: #fdec00;">%LIGHT_INTENSITY%</strong></p>

            </div>
            <div class="col-md-4" >
                <p><a href="/on1"><button class="button">BULB ON</button></a></p>
                <p><a href="/off1"><button class="button button2">BULB
OFF</button></a></p>
            </div>
            <div class="col-md-4">
                <p><a href="/on2"><button class="button">PUMP ON</button></a></p>
```

```
                <p><a href="/off2"><button class="button button2">PUMP
OFF</button></a></p>
            </div>
        </div>
    </div>

      <!-- Bootstrap Bundle with Popper -->
      <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.mi
n.js"></script>
</body>
</html>
```

CSS CODE

```
html {
    font-family: Helvetica;
    display: inline-block;
    margin: 0px auto;
    text-align: center;
    background-position: center;
    transition: background-color 2.5s ease; /* Smooth transition for background
color change */
  }
      /* Style for the credit */
      .credit {
          position: fixed; /* Position it relative to the browser window */
          top: 10px; /* 10px from the bottom */
          left: 10px; /* 10px from the right */
          background-color: rgba(27, 114, 0, 0.562); /* Semi-transparent background
*/
          color: rgb(255, 255, 255);
          padding: 5px 4px;
          border-radius: 5px;
          z-index: 999;
          font-family:  'Dancing Script', cursive;
      }
  .lock-screen, .home-screen {
    position: relative;
    width: auto; /* Adjust width to fit your screen size */
    height: 120vh; /* Adjust height to fit your screen size */
    transition: opacity 0.5s ease;
  }
  .fade-in {
    opacity: 1;
  }
  .wallpaper {
    position: absolute;
```

```
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  object-fit: cover;
}

.lock-content{
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  text-align: center;
}

.lock-content h2 {
  margin-top: 50px;
}

.lock-content input[type="password"] {
  padding: 10px;
  margin: 10px;
  font-size: 16px;
}

.lock-content button {
  padding: 10px 20px;
  font-size: 16px;
  cursor: pointer;
}

.lock-screen {
  position: relative;
}
#pinInput, #unlockBtn {
  margin:auto;

}

.lock-content {
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  text-align: center;
}

.lock-content h2,
```

```
  .lock-content input,
  .lock-content button {
    display: block;
    margin: 0 auto;
  }

  .lock-content input {
    margin-top: 10px; /* Adjust as needed */
  }


  body {
    background-color: white;
    background-image:
url('https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEhocKOdpEDooUakYvGwLEfNU9JYcWHwNIcGXEFWdzC1ESfePzh17GSKggXFAO2wpYgJJUkToqFejQRb0NBoAGYgCNdWte27inuyD7jTg_Gj5Qr4kBgVCx0s2so3hb1ku1OAb6Wydv5ITny0iWtXmZlx0zckJEw0HQd6u48LTik3y7AbzHUGi7tOYsfb_-ri/s1600/4.png');
    background-size: cover;
    background-position: center;
    background-repeat: no-repeat;
    transition: background-color 2.5s ease;
    margin: -10; /* remove default body margin */
    padding: -10; /* remove default body padding */
    height: 100vh; /* set body height to full viewport height */
  }

  .dark-mode {
    background-color: #000000; /* Dark background color */
    background-image:
url('https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEiLoDLZfnoAg54Ql1tAUxuuzk_LG3OGIejW_pTI6jSVS9ypqOaRXF9NJqtj8UfUvgoR4D0m7sQUUHb9klIoZgA6QuLg4jPPhkpRboCXiNomjxu3p4vUdpRWLf5-Y5rXeyXK5xREGLVX9wtrwxG5YXh3HfCY1aizPiKaPBT52GCxXU0xyOIt3JDONAYR0rnd/s1600/3.png');
    background-size: cover;
    background-position: center;
    background-repeat: no-repeat;
    transition: background-color 2.5s ease;
    margin: -10; /* remove default body margin */
    padding: -10; /* remove default body padding */
    height: 100vh; /* set body height to full viewport height */
  }

  h2{
    color: #0F3376;
    padding: 2vh;
    text-shadow: 2px 2px 4px rgb(224, 7, 7); /* Adjust values as needed */
  }
```

```css
p {
  font-size: 1.5rem;
  color: rgb(72, 158, 2);
}

.button {
  display: inline-block;
  background-color: #0300ba;
  border: none;
  border-radius: 4px;
  color: white;
  padding: 16px 40px;
  text-decoration: none;
  font-size: 30px;
  margin: 2px;
  cursor: pointer;
}

.button:hover {
  background-color: #00268e; /* Darken the color on hover */
}

.button2 {
  background-color: #f45c36;
}

.button2:hover {
  background-color: #b22e0a; /* Darken the color on hover */
}
.thermometer {
  width: 70px;
  height: 200px;
  background-color: #ddd;
  border: 2px solid #333;
  position: relative;
  margin-top: 20px;
  overflow: hidden;
  border-radius: 10px;
}

.mercury {
  width: 100%;
  background-color: red; /* Change color as per your design */
  position: absolute;
  bottom: 0;
  transition: height 0.5s;
  border-radius: 0 0 10px 10px;
```

```
}

.thermometer:before {
  content: "";
  display: block;
  width: 10px;
  height: 10px;
  background-color: #333;
  position: absolute;
  top: -6px;
  left: 50%;
  transform: translateX(-50%);
  border-radius: 50%;
}

.thermometer:after {
  content: "";
  display: block;
  width: 6px;
  height: 80px;
  background-color: #333;
  position: absolute;
  top: -90px;
  left: 50%;
  transform: translateX(-50%);
  border-radius: 50% 50% 0 0;
}
.glass {
  width: 100px;
  height: 200px;
  border: 2px solid #000;
  border-radius: 0 0 20% 20%;
  position: relative;
  overflow: hidden;
  margin-top: 20px;
  background-color: #ddd;
}

.water {
  width: 100%;
  background-color: rgb(38, 0, 255); /* Change color as per your design */
  position: absolute;
  bottom: 0;

  transition: height 0.5s;
  border-radius: 0 0 10px 10px;
```

```
}
.ml-mark {
  position: absolute;
  bottom: 0;
  left: 50%;
  transform: translateX(-50%);
  font-size: 10px;
  color: #8802b1;
  font-weight: bold;
}

.scale {
  width: 200px;
  height: 60px;
  border: 2px solid #000;
  border-radius: 20% 20% 20% 20%;
  position: relative;
  overflow: hidden;
  margin-top: 20px;
  background-color: #ddd;
}


.drew {
  height: 100%;
  background-color: rgb(255, 4, 4); /* Change color as per your design */
  position: absolute;
  left: 0;
  right: 35px;
  transition: width 0.5s;
  border-radius: 10px 10px 10px 10px;
}

.bench {
  width: 200px;
  height: 60px;
  border: 2px solid #000;
  border-radius: 20% 20% 20% 20%;
  position: relative;
  overflow: hidden;
  margin-top: 20px;
  background-color: #ddddddaf;
}

.soil {
  height: 100%;
  background-color: rgba(134, 3, 3, 0.37); /* Change color as per your design */
  position: absolute;
```

```
  left: 0;
  right: 120px;
  transition: width 0.5s;
  border-radius: 10px 10px 10px 10px;
}
.line {
  width: 200px;
  height: 60px;
  border: 2px solid #000;
  border-radius: 20% 20% 20% 20%;
  position: relative;
  overflow: hidden;
  margin-top: 20px;
  background-color: #ddddddaf;
}

.tick {
  height: 100%;
  background-color: rgb(4, 112, 27); /* Change color as per your design */
  position: absolute;
  left: 0;
  right: 97px;
  transition: width 0.5s;
  border-radius:10px 10px 10px 10px;
}
.door {
  width: 100px; /* Doubled the width to accommodate left-to-right opening */
  height: 200px;
  background-color: #0e0116;
  position: relative;
  overflow: hidden;
  /* Add your text below */
color: rgb(255, 255, 255); /* Set text color to white */
display: flex; /* Use flexbox for centering */
align-items: center; /* Center vertically */
justify-content: center; /* Center horizontally */
}
.door::before {
content: "Door Closed"; /* Add the text "@foo" before the panel */
font-size: 20px; /* Set the font size */
}

.panel {
width: 100%; /* Set width to 100% */
height: 100%;
background-color: #4ce207;
position: absolute;
top: 0;
```

```
left: -100%; /* Start from the left, outside the door */
transition: transform 0.5s ease-in-out; /* Change the transition property */
transform-origin: left; /* Set transform origin to left */
/* Add your text below */
color: rgb(0, 0, 0); /* Set text color to white */
display: flex; /* Use flexbox for centering */
align-items: center; /* Center vertically */
justify-content: center; /* Center horizontally */
}

.panel::before {
content: "Door Opened"; /* Add the text "@foo" before the panel */
font-size: 20px; /* Set the font size */
}



.open {
  transform: translateX(100%); /* Translate to the right to open the door */
}
```