

The background features abstract, overlapping green geometric shapes in various shades of green, creating a modern and organic feel. The shapes are primarily located on the left and right sides of the frame, framing the central text.

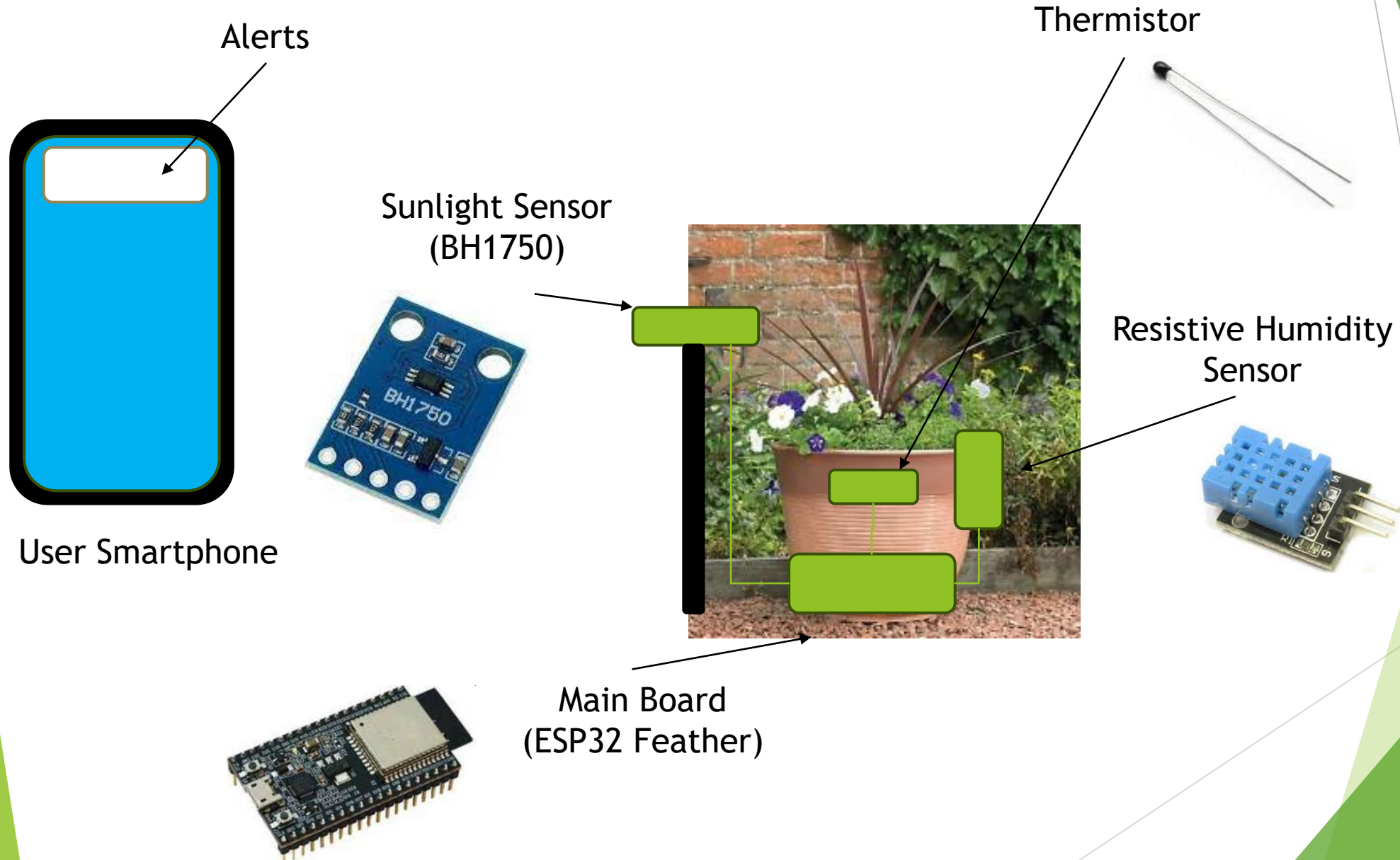
# Smart Plant Pot Design

By Oliver Richardson

# What The Idea Is

- ▶ A 'Smart' plant pot that can tell the user whether the location it is in has enough sunlight and when it needs to be watered.
- ▶ The device collects a selection of data such as:
  - ▶ The current temperature
  - ▶ The current humidity
  - ▶ The current light level
- ▶ It then sends notifications to the user's phone to alert them to any issues with the plant's conditions.

# What The Idea Is



# Power Management

- ▶ The device will put itself into sleep mode which will end when either:
  - ▶ The device is watered, at which point it will send a message to the user's phone before going back to sleep.
  - ▶ It has slept for a hour.
- ▶ This will reduce the power consumption of the device heavily as instead of always being on it will just turn on a few times for a short duration across the day.

# Device Set Up

```
if (return_wakeup_reason() == "Wakeup was not caused by deep sleep: " || preferences.getBool("setup", true)) {  
    Serial.println("Initial Start Up");  
    // get wifi and password from serial line  
    Serial.println("Enter Wifi Name:");  
    while (Serial.available() <= 0) {}  
    SSID = Serial.readString();  
    preferences.putString("SSID", SSID);  
    Serial.println("Enter Wifi Password:");  
    while (Serial.available() <= 0) {}  
    WIFI_PWD = Serial.readString();  
    preferences.putString("WIFI_PWD", WIFI_PWD);  
    // get main variables for the plant  
    Serial.println("How Often Does The Plant Need Watering In Hours:");  
    while (Serial.available() <= 0) {}  
    waterRate = Serial.readString().toInt();  
    preferences.putInt("waterRate", waterRate);  
    waterAlert = waterRate;  
    Serial.println("Ideal Temperature Of Plant In Celsius:");  
    while (Serial.available() <= 0) {}  
    idealTemp = Serial.readString().toInt();  
    preferences.putInt("idealTemp", idealTemp);  
    preferences.putBool("setup", false);  
    preferences.putFloat("average", 0);  
    preferences.putInt("count", 0);  
    delay(2000);  
} else { // load settings  
    SSID = preferences.getString("SSID");  
    WIFI_PWD = preferences.getString("WIFI_PWD");  
    waterRate = preferences.getInt("waterRate");  
    waterAlert = preferences.getInt("waterAlert");  
    idealTemp = preferences.getInt("idealTemp");  
}
```

# Light Meter

```
Wire.begin();
lightMeter.begin(BH1750::ONE_TIME_HIGH_RES_MODE);
// check sensors
// LIGHT
while (!lightMeter.measurementReady(true)) {
  yield();
}
float lux = lightMeter.readLightLevel(); // 500 - bright room
Serial.print("Light: ");
Serial.print(lux);
lightMeter.configure(BH1750::ONE_TIME_HIGH_RES_MODE);
```

# Humidity Sensor

```
preferences.begin("smartPlantPot", false);
average = preferences.getFloat("average");
count = preferences.getInt("count");
preferences.end();
float reading;
reading = readInAnalog(HUMIDITY_PIN);
// convert reading to resistance
reading = (4095 / reading) - 1;
reading = HUMIDITY_RESISTOR_RESISTANCE / reading;
Serial.print("    Humidity Res: ");
Serial.print(reading);
// detect spike in readings
spike = false;
if (initial > 0) {
    average = ((average * count) + reading) / (count + 1);
    count += 1;
    initial -= 1;
} else if ((reading < (average * 1.15)) && (reading > (average * 0.85))) {
    average = ((average * count) + reading) / (count + 1);
    count += 1;
} else {
    spike = true;
}
Serial.print("    Humidity Spike?: ");
Serial.print(spike);
Serial.print("    Humidity Avg: ");
Serial.print(average);
```

```
preferences.putFloat("average", average);
preferences.putInt("count", count);
```

# Thermistor

```
float reading2;  
reading2 = readInAnalog(THERMISTOR_PIN);  
// convert reading to resistance  
reading2 = (4095 / reading2) - 1;  
reading2 = THERMISTOR_RESISTOR_RESISTANCE / reading2;  
Serial.print("    Temperature Res: ");  
Serial.print(reading2);  
// get temperature  
float steinhart;  
steinhart = reading2 / KNOWN_RESISTANCE;  
steinhart = log(steinhart);  
steinhart /= BETA_COEFFICIENT;  
steinhart += 1.0 / (KNOWN_TEMP + 273.15);  
steinhart = 1.0 / steinhart;  
float temperature = steinhart - 273.15;  
Serial.print("    Temperature: ");  
Serial.println(temperature);
```

```
#define KNOWN_TEMP 20.3  
#define KNOWN_RESISTANCE 6200  
#define BETA_COEFFICIENT 26367.1  
#define THERMISTOR_PIN A1  
#define THERMISTOR_RESISTOR_RESISTANCE 1000
```



# Flags

```
if(temperature > idealTemp + 3 || temperature < idealTemp - 3 )
{
    pflags |= PFLAG_TEMP;
}
if(lux < 50)
{
    pflags |= PFLAG_LIGHT;
}
if(spike)
{
    pflags |= PFLAG_WATER;
}
if(waterAlert == 0)
{
    waterAlert = waterRate;
    pflags |= PFLAG_WATER;
}
```

```
attachInterrupt(digitalPinToInterrupt(26),watered,RISING); // if watered run watered()
```

```
void watered(){
    // if humidity spike then trip flag
    if(spike)
    {
        pflags |= PFLAG_WATER;
    }
}
```

```
if(return_wakeup_reason() == "Wakeup caused by timer")
{
    preferences.putInt("waterAlert", waterAlert-1);
}
```

# MQTT Set Up

```
// check flags
if (pflags) {
    // connect to the wifi
    Serial.print("Connecting to ");
    Serial.print(SSID);
    WiFi.begin(SSID, WIFI_PWD);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(" . ");
    }
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
    // connect to MQTT
    client.setServer(MQTT_BROKER, MQTT_PORT);
    while (!client.connected()) {
        if (client.connect(("ESP32-" + String(random(0xffff), HEX)).c_str())) {
            Serial.println("MQTT connected.");
        } else {
            Serial.printf(" failed , rc=%d try again in 5 seconds", client.state());
            delay(5000);
        }
    }
    // subscribe to topic and set callback function to call when msg arrives
    client.subscribe(MQTT_SUBSCRIBE_TOPIC);
    client.setCallback(manualActivation);
}
```

# MQTT Messages

```
noInterrupts();
uint8_t cflags = pflags;
pflags = 0x00;
interrupts();
if (cflags & PFLAG_WATER) {
    lastwateredTime = millis();
    nextwateredTime = lastwateredTime + (waterRate * hoursToMillis);
    String watermsg = "The Plant has been watered, water it again in ";
    watermsg += waterRate;
    watermsg += " hours.";
    client.publish(MQTT_PUBLIC_TOPIC, watermsg.c_str());
}
if (cflags & PFLAG_TEMP) {
    client.publish(MQTT_PUBLIC_TOPIC, "The Plant's Temperature is +/- 3 degrees celsius from it's ideal temperature, it would be a good idea to move it.");
}
if (cflags & PFLAG_LIGHT) {
    client.publish(MQTT_PUBLIC_TOPIC, "The Plant is in the dark, if it is not night then you might want to move it.");
}
}
```

# Deep Sleep

```
String return_wakeup_reason() {  
    esp_sleep_wakeup_cause_t wakeup_reason;  
  
    wakeup_reason = esp_sleep_get_wakeup_cause();  
  
    switch (wakeup_reason) {  
        case ESP_SLEEP_WAKEUP_EXT0: return "Wakeup caused by external signal using RTC_IO";  
        case ESP_SLEEP_WAKEUP_EXT1: return "Wakeup caused by external signal using RTC_CNTL";  
        case ESP_SLEEP_WAKEUP_TIMER: return "Wakeup caused by timer";  
        case ESP_SLEEP_WAKEUP_TOUCHPAD: return "Wakeup caused by touchpad";  
        case ESP_SLEEP_WAKEUP_ULP: return "Wakeup caused by ULP program";  
        default: return "Wakeup was not caused by deep sleep: " + wakeup_reason;  
    }  
}
```

```
esp_sleep_enable_ext0_wakeup(GPIO_NUM_26,1); // wake up if watered
```

```
esp_sleep_enable_timer_wakeup(60 * 60 * uS_TO_S_FACTOR);  
// wait  
esp_deep_sleep_start();
```

# Further Potential

- ▶ Could Set up an automated system with servo motors that waters the device when it receives the MQTT messages.
- ▶ Could integrate a water API to avoid sending messages when it is going to rain and alert to user that temperature is going to change in advance.