TABLE OF CONTENTS

Smart Irrigation System Final Report

Project Introduction & Proposal etc:

Problem:

Modern agricultural and gardening practices, whether in households or commercial environments, have to consistently deal with the challenges of achieving optimal watering. This not only pertains to the frequency and amount but also considers when and how based on real-time plant and environmental needs. Manual watering, as seen traditionally, can lead to inconsistent watering due to irregularity, resulting in uneven water distribution. This, coupled with human error, leads to problems of overwatering or underwatering, compromising plant health and causing unnecessary water wastage. Moreover, without a continuous monitoring system in place, it becomes a challenge to accurately assess and address real-time water requirements of plants.

Objective:

The primary goal of this project was to develop a smart crop irrigation system, dubbed the Smart Crop Irrigation System, that not only addresses the irregularities and inefficiencies of manual watering but also utilizes real-time data to optimize water usage. Using a combination of various sensors and a microcontroller, this system aimed to continually monitor crucial factors such as soil moisture levels, temperature and humidity of the environment and light intensity. Based on this data, the irrigation system would be better equipped to determine the appropriate timing and quantity of water required for optimal plant growth and health, while simultaneously promoting water conservation.

Product Major Aspects:

Sensors Integration:

The Smart Crop Irrigation System incorporates various sensors to measure vital environmental and soil factors. These sensors include:

- **Soil Moisture Sensor:** Directly measures the moisture content in the soil, enabling the system to determine when the plants need water. *(Referenced from: Smith, J.M., & Bristow, K.L. (2002). Measurement of soil water content using a simplified impedance measuring technique. Journal of Agricultural and Forest Meteorology, 111, 183-194.)*
- **Temperature Sensor:** Monitors ambient temperature, which plays a vital role in plant metabolism and evapotranspiration rates.
- **Light Sensor:** Measures the light intensity, ensuring plants receive adequate light for photosynthesis.
- **Humidity Sensor:** This determines the ambient humidity, which can affect plant transpiration and growth.

Microcontroller (ESP2668):

The core of the system is the ESP2668 microcontroller, which collects and processes data from the sensors. The logic embedded within the microcontroller evaluates the sensor inputs against the set optimal values and then decides whether the plant requires watering. The choice of ESP8266 was made due to its capability to provide smart features while maintaining functionality in unreliable internet conditions. Its compatibility with various sensors also made it a prime candidate for this system.

Automated Watering Mechanism:

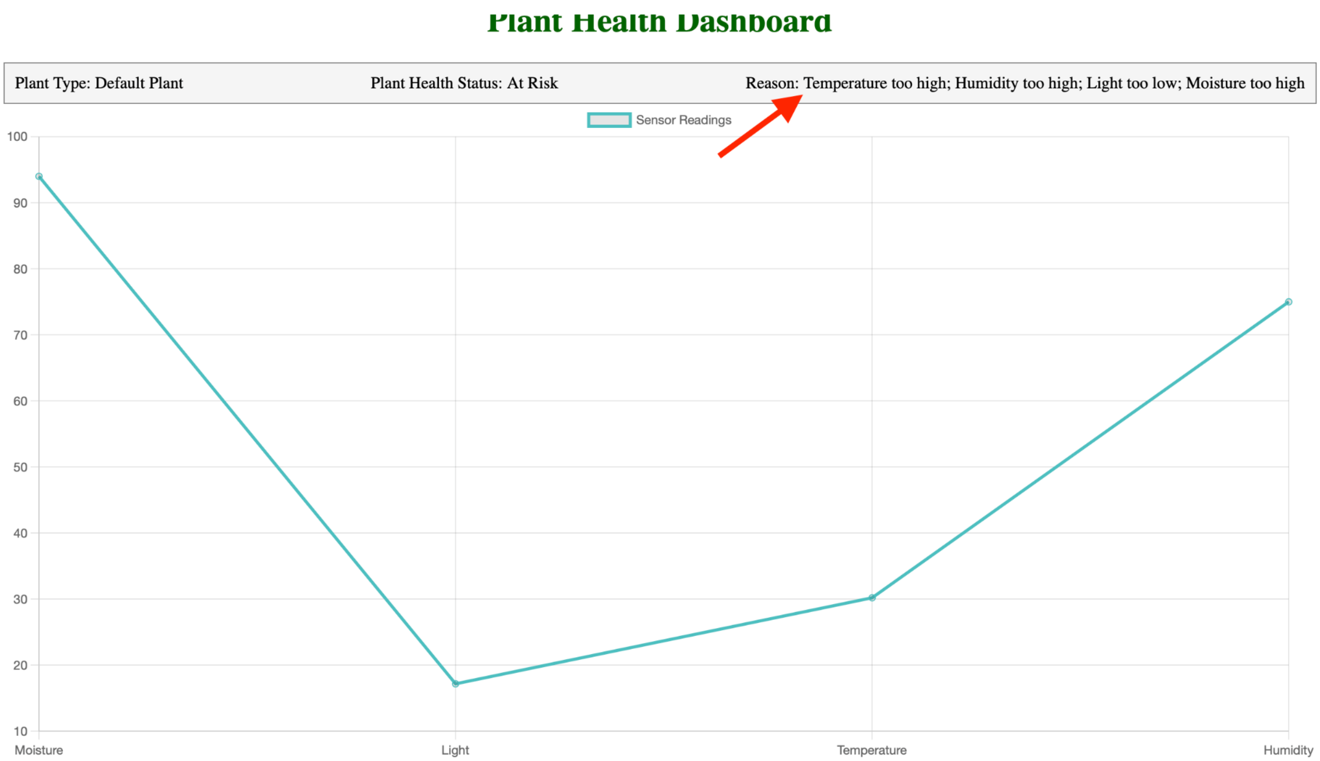
Based on the microcontroller's decision, the pump and the associated actuator mechanism is activated to deliver the precise amount of water needed. This aspect of the product eliminates manual intervention, ensuring that plants receive water exactly when they need it, in the right amounts. *(Referenced from: Uddin, J.M., Smith, R.J., & Gillies, M.H. (2019). Evaluating automated irrigation systems for horticulture. Irrigation Science, 37, 331-342.)*

Plant Health Dashboard:

The user interface displays crucial information, including the current values from the sensors and the health status of the plant. Depending on the plant type chosen, the system evaluates the current conditions against optimal values and provides a health status. This feature ensures that users can take corrective actions, if needed, in a timely manner.

Decision Logic for Plant Health:

The decision-making logic incorporated into the software evaluates sensor readings against ideal conditions. The thresholds, such as 18-24°C for temperature and 40-65% for humidity, light intensity of 60% or more , and soil moisture level between 50 - 80% serve as the basis for these decisions. Any deviation beyond these thresholds triggers an 'at risk' alert on the dashboard, highlighting the specific parameter or parameters causing the risk.



Rationale for Design and Implementation Decisions:

Choice of Sensors: The selection of specific sensors was primarily based on the most influential environmental and soil factors that impact plant health. Their inclusion ensures comprehensive monitoring of plant conditions.
Use of ESP2668: This microcontroller was chosen due to its versatility, ease of integration with sensors, and ability to function in varied internet conditions.
Integration of Plant Health Dashboard: Providing users with a real-time visual representation of plant health and environmental conditions ensures proactive care. It gives a clear, immediate feedback mechanism to users, bridging the gap between advanced technology and user-friendly interfacing. This dashboard is available via [ESP8266 ip address]/dashboard and REST endpoint that responds with JSON data is available at [ESP8266 ip address]/data. For my Dashboard, they are available at http://192.168.100.214/dashboard and http://192.168.100.214/data


```
{"moisture": 93.00,"light": 17.08,"temperature": 30.20,"humidity": 75.00,"plantType": "Default Plant","plantHealthStatus": "At Risk","plantHealthReason": "Temperature too high; Humidity too high; Light too low; Moisture too high"}
```


Setting Optimal Values: The chosen range for various parameters was based on extensive literature suggesting ideal conditions for most plant types. These values serve as the guiding principle for the system's decision-making logic. *(Referenced from: Taiz, L., Zeiger, E., Møller, I.M., & Murphy, A. (2018). Plant Physiology and Development (6th ed.). Sinauer Associates.)*

The Smart Crop Irrigation System serves as an innovative solution to traditional challenges faced in plant care. Through its smart integration of technology and user-centric design, it offers a promising way forward in ensuring plant health while promoting sustainable water use practices.

Description of the Development Lifecycle

The software development lifecycle used for this project was the **Agile Development Methodology**. This methodology was chosen due to its iterative approach, which is conducive for projects that require continuous changes and adaptability.

Requirement Gathering:

Initially, the requirements of the project were identified, namely the environmental factors that needed to be monitored like moisture, light, temperature, and humidity, and the plant health dashboard.

Planning:

Here, it was decided which sensors to use, which microcontroller would be best suited for the task, and the integration of all components into a functional system.

Implementation and Design:

To develop an effective plant health monitoring system, various sensors and modules are integrated to work together. The NodeMCU, a development board based on the ESP8266, serves as the central processing unit. Below is a detailed breakdown of the system design and wiring connections:

BH1750 Light Sensor:

This sensor measures the ambient light intensity, which is crucial for assessing plant health. Here is how it's wired to the NodeMCU:

- VCC of BH1750 to 3.3V of NodeMCU.
- GND of BH1750 to GND of NodeMCU.
- SDA of BH1750 to D6 of NodeMCU.
- SCL of BH1750 to D5 of NodeMCU.

OLED Display:

The OLED provides a real-time display of readings from the sensors.

- SDA of OLED to the SDA (D2) of NodeMCU.
- SCL of OLED to the SCL (D1) of NodeMCU.
- VCC of OLED to 3.3V of NodeMCU.
- GND of OLED to GND of NodeMCU.

Soil Moisture Sensor:

This sensor checks the moisture content in the soil, crucial for determining watering needs.

- VCC: Connect to the 3.3V pin of the NodeMCU.
- GND: Connect to any GND (Ground) pin on the NodeMCU.
- OUT: Connect to A0 on the NodeMCU.

DHT11 Temperature and Humidity Sensor:

Both the VCC of the DHT11 and the NodeMCU's 3v3 are connected to the power rails on the breadboard, while the GND of the DHT11 and the NodeMCU's GND are connected to the ground rail. Specifically:

- VCC: Connect to the 3.3V rail on the breadboard.
- GND: Connect to the GND rail on the breadboard.
- Data: Connect to the D3 pin on the NodeMCU. Ensure there's a 10kΩ resistor between the Data pin of the DHT11 and the 3.3V rail on the breadboard.

Relay and Pump Wiring:

The relay controls the pump, ensuring that it turns on/off based on moisture conditions.

- Transistor (S8550):
  - Emitter to ground rail on the breadboard.
  - Collector to the "IN" pin of the relay module.
  - Base connected via a 1kΩ resistor to the D7 pin of the ESP8266 and to the VCC rail on the breadboard through a 10kΩ resistor.
- Relay Module:
  - VCC to the VCC rail on the breadboard.
  - GND to the ground rail on the breadboard.
  - IN to the collector of the S8550 transistor.
- Pump:
  - Red Wire to the Normally Open (NO) terminal of the relay.
  - Black Wire directly to the negative (-) terminal of the power supply.
- External Power Supply:
  - Positive to the Common (COM) terminal of the relay.
  - Ground to the ground rail on the breadboard.

Software Design and Implementation:

The design phase was about how to structure the code. Clear distinctions were made between the different functionalities such as reading sensor data, managing relay, the plant health status logic, and the dashboard web interface.
For the ease of modularity, and future scalability, functions such as updatePlantHealth, getDataAsJSON, and setupServerRoutes were separated out, so they can be easily modified, extended or replaced in future iterations.
After designing the system, the implementation began. The core of the system is the loop function, which continually reads data from sensors, updates plant health status, and manages the relay accordingly. The code initializes and reads values from the BH1750 light sensor, the soil moisture sensor, and the DHT11 sensor. It then processes the data to determine the plant's health, updating the OLED display with relevant metrics. Additionally, the relay is controlled based on moisture and temperature conditions. Furthermore, an HTTP server provides a web dashboard for remotely viewing the plant's health metrics. With the above design and implementation, you can effectively monitor and manage the health of your plant by observing various environmental parameters and controlling water supply as necessary.

Validation:

This involved running the system and ensuring that all parts work together as intended. When the sensors read values below or above the optimal range, the system correctly identified the health of the plant as "At Risk" and provided the appropriate reasons.

Verification:

To ensure accuracy, the readings of the sensors were cross-verified with manual measurements. The relay operations were also monitored to ensure they triggered under the defined conditions.

Iterative Testing and Feedback:

Using Agile's iterative approach, after each significant change or addition to the code, the system was tested. This ensured immediate feedback and allowed for timely corrections. This approach proved invaluable, especially when integrating the web dashboard, which had to seamlessly fetch the latest sensor data.

Deployment:

After validation and verification, the system was deployed in an actual environment. The ESP8266, connected to the WiFi, served the dashboard on the network, allowing real-time monitoring of plant health.

Maintenance:

Due to the agile nature of the development, changes can be easily made in the future. Whether it's incorporating more sensors, changing the triggering conditions, or even updating the web interface, the modular design ensures easy maintenance and upgradation.

Throughout this lifecycle, the project heavily leaned on academic and industry resources for the appropriate usage of sensors, the ESP microcontroller, and best practices for software development. Key references include the documentation for libraries used like U8g2lib, DHT, ESPAsyncWebServer, and others. These resources were pivotal in ensuring the system's robustness and efficiency.

Critical Analysis and Reflection

What Went Right

Despite the challenges I faced, such as the faulty IC2 LCD module, I was able to pivot and incorporate an OLED screen into the project. This adaptability ensured the project continued despite unforeseen complications. The consistent issues with the DHT module's signal and the 5V relay presented challenges that tested my troubleshooting abilities. Ultimately, I was able to identify the solutions — adding a 10k pull-up resistor and incorporating a transistor and resistor respectively. Due to the transition from the IC2 LCD to the OLED screen, I made the necessary adjustments to the codebase, showing my flexibility in adapting to hardware changes.

What Went Wrong

The faulty IC2 LCD module highlighted a need for a more thorough preliminary testing phase for all components before integration into the project. While I eventually addressed the DHT module's signaling issue and the relay problem, anticipating common issues associated with these components might have saved me valuable troubleshooting time.

What Could Be Done Differently Next Time

- Pre-project Component Testing - Before commencing on the main project, a rigorous test of all individual components can be beneficial in early identification of faulty modules.
- Research on Common Component Issues - A preparatory phase where I research common problems associated with the components I'm using could help in faster problem resolution.
- Documentation - Maintaining detailed documentation during the troubleshooting phase could serve as a reference for future projects, helping to avoid repeating the same issues.

Appraisal of the Product

The final product, despite its initial hiccups, stands as a testament to perseverance and adaptability. Although there were component setbacks, the project showcased the importance of problem-solving in the realm of electronics.

Analysis of the Approach Taken with Hindsight

In hindsight, while my approach was mostly reactive, I demonstrated resilience. Future projects would benefit from a proactive strategy where potential pitfalls are anticipated and addressed in the planning phase.

Analysis of Software/Tools Used

The software and libraries used for the OLED display and DHT module were integral in ensuring the project's success. Adapting to a different screen and its associated library presented a learning curve but expanded my skill set. Future projects could see me exploring tools that provide diagnostics or simulations before actual implementation, reducing the troubleshooting phase.By reflecting on these aspects, not only do I appreciate the journey of this project but I also equip myself with the knowledge to navigate future endeavors with increased efficiency and foresight.

References

1. *The Best Smart Sprinkler Controllers of 2023. (2023, June 21). Reviewed.* *https://reviewed.usatoday.com/smarthome/best-right-now/the-best-smart-sprinkler-controllers*
2. *150+ ESP8266 NodeMCU Projects, Tutorials and Guides with Arduino IDE | Random Nerd Tutorials*. (2021, February 3). Random Nerd Tutorials. https://randomnerdtutorials.com/projects-esp8266/
3. A. (2021, June 5). *IoT based Smart Irrigation | Soil-Moisture | DHT11 | NodeMCU*. Techatronic. https://techatronic.com/iot-based-smart-irrigation/
4. *Nxeco - Smart Weather-based irrigation Controller*. (n.d.). Nxeco - Smart Weather-based Irrigation Controller. http://www.nxeco.com
5. *What are the Advantages of a Smart Irrigation System?* (2022, January 10). HydroPoint. https://www.hydropoint.com/blog/what-are-the-advantages-of-a-smart-irrigation-system
6. *Rachio 3 Smart Sprinkler Controller and Smart Water System FAQ*. (n.d.). Rachio 3 Smart Sprinkler Controller and Smart Water System FAQ. https://support.rachio.com/en_us/rachio-3-smart-sprinkler-controller-and-smart-water-system-faq-S1vIwLJKD#:~:text=Rachio%203%20includes%20on%2Dunit,web%20app%20from%20any%20computer.
7. *Smart Irrigation Technology: Controllers and Sensors - Oklahoma State University*. (2017, February 1). Smart Irrigation Technology: Controllers and Sensors | Oklahoma State University. https://extension.okstate.edu/fact-sheets/smart-irrigation-technology-controllers-and-sensors.html

Appendices

Screenshots of the Final Source Code

```
#include <U8g2lib.h>
#include <Wire.h>
#include <BH1750.h>
#include <DHT.h>
#include <ESP8266WiFi.h>
#include <ESPAsyncTCP.h>
#include <ESPAsyncWebSrv.h>

// OLED setup
U8G2_SSD1306_128X64_NONAME_F_SW_I2C u8g2(U8G2_R0, SCL, SDA, U8X8_PIN_NONE);

// Light sensor
BH1750 lightMeter;

// Moisture sensor
#define MOISTURE_PIN A0
#define RELAY_PIN D7
float moisturePercentage = 0.0;
// Additional global variables
float lightPercentage = 0.0;
float temperature = 0.0;
float humidity = 0.0;

//Global variables for plant health
String plantType = "Default Plant";
String plantHealthStatus = "Good";
String plantHealthReason = "";

void updatePlantHealth() {
  plantHealthStatus = "Good";
  plantHealthReason = "";

```

```
  if (temperature < 18 || temperature > 24) {
    plantHealthStatus = "At Risk";
    plantHealthReason += temperature < 18 ? "Temperature too low; " : "Temperature too high; ";
  }

  if (humidity < 40 || humidity > 65) {
    plantHealthStatus = "At Risk";
    plantHealthReason += humidity < 40 ? "Humidity too low; " : "Humidity too high; ";
  }

  if (lightPercentage < 60 || lightPercentage > 80) {
    plantHealthStatus = "At Risk";
    plantHealthReason += lightPercentage < 60 ? "Light too low; " : "Light too high; ";
  }

  if (moisturePercentage < 50 || moisturePercentage > 80) {
    plantHealthStatus = "At Risk";
    plantHealthReason += moisturePercentage < 50 ? "Moisture too low" : "Moisture too high";
  }
}

// DHT sensor
#define DHT_PIN D3
#define DHT_TYPE DHT11
DHT dht(DHT_PIN, DHT_TYPE);

// WiFi Credentials
const int MAX_WIFI_RETRIES = 10;  // maximum number of times to try connecting to WiFi
const int WIFI_RETRY_DELAY = 1000;  // delay between each retry in milliseconds

const char* ssid = "Digicel_WiFi_gVdj";
const char* password = "kPT9sAWB";
```

```
      delay(WIFI_RETRY_DELAY);
      Serial.println("Connecting to WiFi...");
      wifiRetries++;
    }

    if(WiFi.status() == WL_CONNECTED) {
      Serial.println("Connected to WiFi");
      Serial.print("IP Address: ");
      Serial.println(WiFi.localIP());
    } else {
      Serial.println("Failed to connect to WiFi after multiple attempts");
      // Optional: Take an appropriate action here (like going to deep sleep, rebooting, etc.)
    }



  setupServerRoutes();  // Setup the web server routes
}

void loop() {
  // Read moisture sensor
  int rawMoisture = analogRead(MOISTURE_PIN);
  moisturePercentage = map(rawMoisture, 405, 870, 100, 0);

  // Read light sensor
  float lightIntensity = lightMeter.readLightLevel();
  lightPercentage = (lightIntensity / 1000.0) * 100.0;
  if (lightPercentage > 100) lightPercentage = 100;  // cap it at 100%

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

```
AsyncWebServer server(80);  // Create a web server on port 80

// Forward declaration
void setupServerRoutes();

void setup() {
    // Moisture Sensor
    pinMode(RELAY_PIN, OUTPUT);
    digitalWrite(RELAY_PIN, HIGH);  // Initially keep the relay off

    // OLED Initialization
    u8g2.begin();
    u8g2.setFont(u8g2_font_ncenB12_tr);

    // Light Sensor Initialization
    Wire.begin(D6, D5); // D6 is SDA, D5 is SCL
    lightMeter.begin();

    // DHT Sensor Initialization
    dht.begin();
    delay(2000);

    // Serial Initialization
    Serial.begin(9600);

    // Connect to WiFi
    WiFi.begin(ssid, password);

    int wifiRetries = 0;
    while (WiFi.status() != WL_CONNECTED && wifiRetries < MAX_WIFI_RETRIES) {
      delay(WIFI_RETRY_DELAY);
```

```
  // Reports plant health status
  updatePlantHealth();


  // Control the relay
  if (moisturePercentage <= 50.00 || temperature > 32.0) {
    digitalWrite(RELAY_PIN, LOW); // Turn ON
  } else if (moisturePercentage >= 80.00) {
    digitalWrite(RELAY_PIN, HIGH);  // Turn OFF
  }

  // Display values on OLED
  u8g2.clearBuffer();
  u8g2.drawStr(0, 15, ("Mois: " + String(moisturePercentage) + "%").c_str());
  u8g2.drawStr(0, 30, ("Light: " + String(lightPercentage) + "%").c_str());
  u8g2.drawStr(0, 45, ("Temp: " + String(temperature) + "C").c_str());
  u8g2.drawStr(0, 60, ("Hum: " + String(humidity) + "%").c_str());
  u8g2.sendBuffer();

  delay(1000);
}

String getDataAsJSON() {
  updatePlantHealth();
  String data = "{";
  data += "\"moisture\": " + String(moisturePercentage) + ",";
  data += "\"light\": " + String(lightPercentage) + ",";
  data += "\"temperature\": " + String(temperature) + ",";
  data += "\"humidity\": " + String(humidity) + ",";
  data += "\"plantType\": \"" + plantType + "\",";
  data += "\"plantHealthStatus\": \"" + plantHealthStatus + "\",";
  data += "\"plantHealthReason\": \"" + plantHealthReason + "\"";
```

```
        data += "}";
        return data;
}


String getDashboardHTML() {
        return R"html(
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Plant Health Dashboard</title>
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <style>
        table, th, td {
            border: 1px solid black;
            border-collapse: collapse;
        }
        h1 {
            text-align: center;
            color: darkgreen;
        }
        #healthStatus {
            display: flex;
            justify-content: space-between;
            margin-top: 20px;
            padding: 10px;
            border: 1px solid gray;
            background-color: #f5f5f5;
        }
        .good {
```

```
            .good {
                color: green;
            }
            .atRisk {
                color: red;
            }
        </style>
</head>
<body>
    <h1>Plant Health Dashboard</h1>
    <div id="healthStatus">
        <div>Plant Type: <span id="plantTypeValue"></span></div>
        <div>Plant Health Status: <span id="plantHealthStatusValue" class=""></span></div>
        <div>Reason: <span id="plantHealthReasonValue"></span></div>
    </div>
    <canvas id="dataChart" width="400" height="200"></canvas>
    <table id="dataTable">
        <thead>
            <tr>
                <th>Parameter</th>
                <th>Value</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>Moisture (%)</td>
                <td id="moistureValue"></td>
            </tr>
            <tr>
                <td>Light (%)</td>
                <td id="lightValue"></td>
            </tr>
```

```
        <tr>
            <td>Temperature (°C)</td>
            <td id="temperatureValue"></td>
        </tr>
        <tr>
            <td>Humidity (%)</td>
            <td id="humidityValue"></td>
        </tr>
    </tbody>
</table>
<script>
    )html" "function fetchDataAndPopulate() {" R"html(
        fetch('/data')
            .then(response => response.json())
            .then(data => {
                // Populate plant health status
                document.getElementById('plantTypeValue').innerText = data.plantType;
                const healthStatusElem = document.getElementById('plantHealthStatusValue');
                healthStatusElem.innerText = data.plantHealthStatus;
                healthStatusElem.className = data.plantHealthStatus.toLowerCase();
                document.getElementById('plantHealthReasonValue').innerText = data.plantHealthReason;

                // Populate chart
                const ctx = document.getElementById('dataChart').getContext('2d');
                new Chart(ctx, {
                    type: 'line',
                    data: {
                        labels: ['Moisture', 'Light', 'Temperature', 'Humidity'],
                        datasets: [{
                            label: 'Sensor Readings',
                            data: [data.moisture, data.light, data.temperature, data.humidity],
                            borderColor: 'rgba(75, 192, 192, 1)',
```

```
                    document.getElementById('temperatureValue').innerText = data.temperature + "°C";
                    document.getElementById('humidityValue').innerText = data.humidity + "%";
                });
            }

            fetchDataAndPopulate();
            setInterval(fetchDataAndPopulate, 10000);
        </script>
</body>
</html>
)html";
}



void setupServerRoutes() {
        // Endpoint to serve the dashboard
    server.on("/dashboard", HTTP_GET, [](AsyncWebServerRequest* request) {
        request->send(200, "text/html", getDashboardHTML());
    });

    // Endpoint to serve data as JSON
    server.on("/data", HTTP_GET, [](AsyncWebServerRequest* request) {
        request->send(200, "application/json", getDataAsJSON());
    });

    server.begin();  // Start the server
}
```

Instructions on How to Develop the Physical Computing Circuit:

1. Wiring BH1750 Light Sensor to NodeMCU:

- **Step 1.1:** Connect the VCC of the BH1750 to the 3.3V pin of the NodeMCU.
- **Step 1.2:** Connect the GND of BH1750 to the GND pin of the NodeMCU.
- **Step 1.3:** Connect the SDA pin of BH1750 to D6 of the NodeMCU.
- **Step 1.4:** Connect the SCL pin of BH1750 to D5 of the NodeMCU.

2. Wiring OLED Display:

- **Step 2.1:** Connect the SDA pin of OLED to D2 of the NodeMCU.
- **Step 2.2:** Connect the SCL pin of OLED to D1 of the NodeMCU.
- **Step 2.3:** Connect the VCC of OLED to the 3.3V pin of NodeMCU.
- **Step 2.4:** Connect the GND of OLED to any GND pin of the NodeMCU.

3. Wiring Soil Moisture Sensor:

- **Step 3.1:** Connect the VCC pin of the soil moisture sensor to the 3.3V pin of the NodeMCU.
- **Step 3.2:** Connect the GND pin of the soil moisture sensor to any GND pin of the NodeMCU.
- **Step 3.3:** Connect the OUT or signal pin of the soil moisture sensor to the A0 pin of the NodeMCU.

4. Wiring Relay with Transistor and NodeMCU:

- **Step 4.1:** Connect the emitter of the S8550 transistor to the GND rail on the breadboard.
- **Step 4.2:** Connect the collector of the transistor to the "IN" pin of the relay module.
- **Step 4.3:** Connect the base of the transistor to one leg of a 1kΩ resistor, then connect the other leg to the D7 pin of the NodeMCU. Additionally, connect the base of the transistor to the 3.3V rail on the breadboard using a 10kΩ resistor.
- **Step 4.4:** Connect the VCC pin of the relay module to the 3.3V rail on the breadboard.
- **Step 4.5:** Connect the GND pin of the relay module to the GND rail on the breadboard.

5. Wiring Water Pump with Relay:

- **Step 5.1:** Connect the red wire of the pump (Positive) to the Normally Open (NO) terminal of the relay.

- **Step 5.2:** Connect the black wire of the pump (Negative) to the negative (-) terminal of your power supply.
- **Step 5.3:** Connect the positive (+) terminal of your power supply (that matches the pump's voltage requirements) to the Common (COM) terminal of the relay.
- **Step 5.4:** Connect the negative (-) terminal of the power supply to the common GND rail on the breadboard.

6. Wiring DHT11 Temperature and Humidity Sensor:

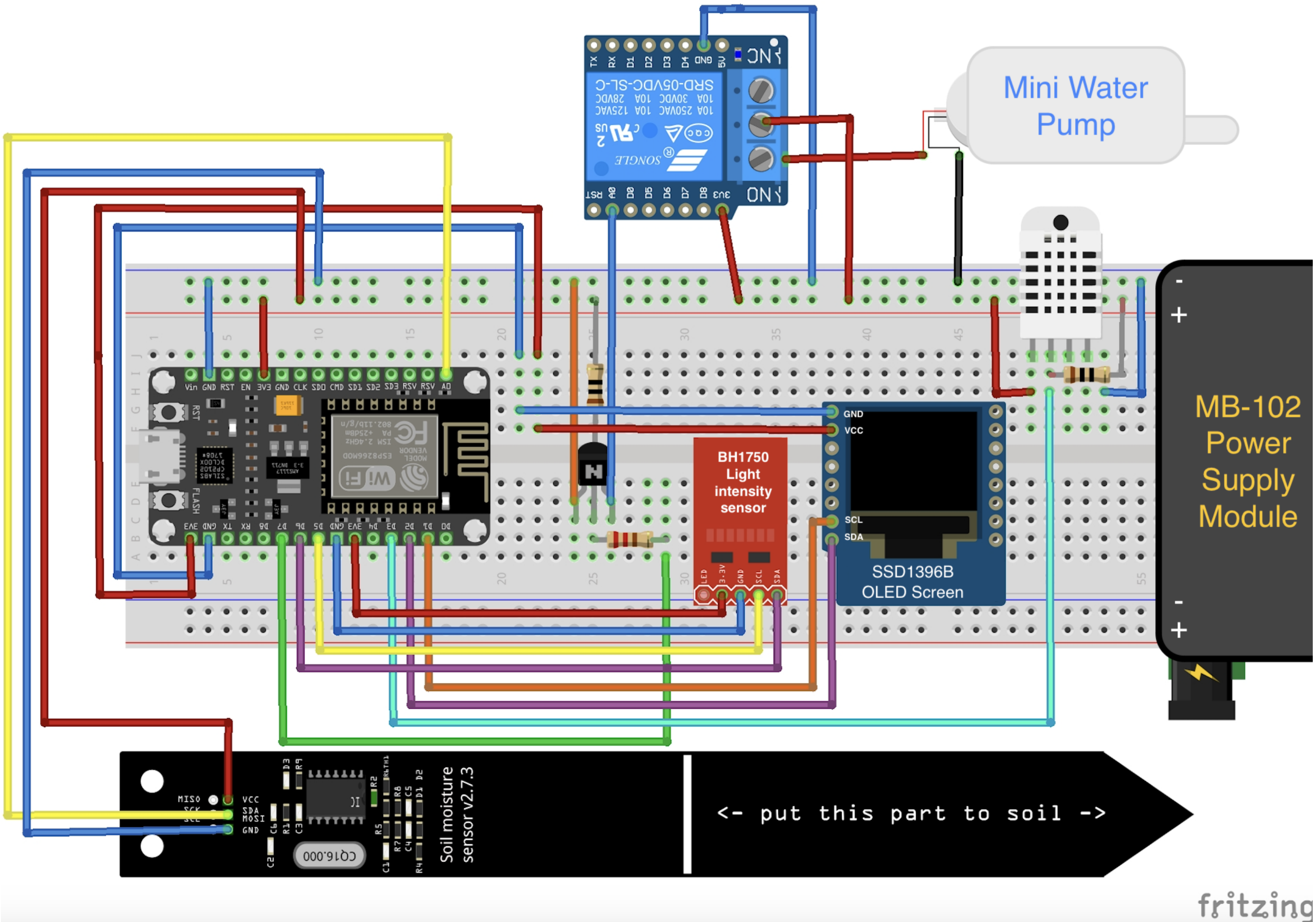
- **Step 6.1:** Connect the VCC pin of the DHT11 to the 3.3V rail on the breadboard.
- **Step 6.2:** Connect the GND pin of the DHT11 to the GND rail on the breadboard.
- **Step 6.3:** Connect the Data/Out pin of the DHT11 to the D3 pin on the NodeMCU.
- **Step 6.4:** Connect a 10kΩ resistor between the Data/Out pin of the DHT11 and the 3.3V rail on the breadboard (acting as a pull-up resistor).

Note: Ensure the NodeMCU's 3V3 pin is connected to the 3.3V rail on the breadboard, and its GND pin is connected to the GND rail on the breadboard for a consistent power supply throughout the components.

7. Setting up the Breadboard Power Module (MB-102) + 9v adapter:

- **Step 7.1:** Position the MB-102 breadboard power module so that its pins align with the power and ground rails on the breadboard.
- **Step 7.2:** Carefully insert the pins of the power module into the power and ground rails of the breadboard. Ensure they are securely connected
- **Step 7.3:** Locate the barrel jack on the breadboard power module.
- **Step 7.4:** Plug the 9V adapter into the barrel jack of the breadboard power module.
- **Step 7.5:** Check the voltage selection jumper or switch on the breadboard power module. Most modules will allow you to choose between 3.3V and 5V. Ensure it is set to the voltage level appropriate for your components (for this setup, select 3.3V).
- **Step 7.6:** Once the correct voltage is selected, plug the 9V adapter into a wall outlet or power strip. The power module should have LEDs indicating power status; ensure they are on and displaying the correct colors (typically green or red).
- **Step 7.7:** Your breadboard should now be powered. You can use the positive (+) and ground (-) rails on the breadboard to distribute power and ground to your components.

Circuit Drawing



DHT 11	
VCC	+
GND	-
DATA	D3 NodeMCU
BH1750 Light intensity sensor	
VCC	3.3v
GND	GND
SDA	D6 NodeMCU
SCL	D5 NodeMCU
SSD1396B OLED Display	
VCC	3.3v
GND	GND
SDA	D2 NodeMCU
SCL	D1 NodeMCU
Soil Moisture Sensor	
VCC	+
GND	-
A0	A0 NodeMCU
5V Relay	
VCC	+
GND	-
IN	D7
COM	-
Mini Pump	
Red Wire	NO
Black Wire	-

Components List

1. ESP8266 NodeMCU
2. DHT11 Humidity/ Temperature sensor module
3. BH1750 Light intensity sensor
4. SSD1396B OLED Display
5. Capacitive Soil Moisture Sensor
6. 1 Channel 5V Relay Module
7. Mini water pump
8. Breadboard
9. Breadboard power module+MB-102 830
10. Male jumper wires
11. Female jumper wires
12. Female to male jumper wires
13. 10k resistor
14. 220k resistor
15. 1k resistor
16. Transistor (S8550)
17. 9v adapter
18. Vinyl Tubing