**Smart Plant Monitoring and Care System**

**1. Objectif du projet**

Développer un système automatisé pour surveiller et prendre soin des plantes en fonction des conditions environnementales détectées, telles que la lumière, la température, l'humidité et l'humidité du sol, et permettre une intervention manuelle à travers une interface utilisateur

**2. Fonctionnalités du système**

**Surveillance environnementale** : Mesure continue de la température, de l'humidité, de la lumière, et de l'humidité du sol.

**Contrôle automatique de l'arrosage** : Activation d'un système d'arrosage basé sur le niveau d'humidité du sol détecté.

**Indicateurs de santé de la plante** : Utilisation de LED RGB pour afficher l'état de santé de la plante.

**Alertes et notifications** : Utilisation d'un buzzer pour les alertes critiques et modification des couleurs de la LED RGB.

**Interface de contrôle manuel** : Boutons physiques et télécommande IR pour l'interaction utilisateur.

**Sécurité** : Surveillance des niveaux d'eau et alertes pour éviter le dessèchement du réservoir.

**3. Composants matériels**

***Capteurs :***

Capteur de température et d'humidité

Photorésistance

Capteur de niveau d’eau (mesure & pluie)

**Capteur humidité au sol (ou similaire)**

**Actuateurs :**

**Pompe à eau**

Relais (pour contrôler la pompe à eau)

Buzzer (pour les alertes sonores)

**Indicateurs :**

LED RGB (pour l'indication de l'état de santé)

**Contrôle :**

Bouton

Récepteur IR (pour la télécommande)

Microcontrôleur :

Par exemple, Arduino, ESP32, ou Raspberry Pi (selon la complexité du traitement requis)

**4. Exigences logicielles**

**Firmware :**

Lecture et interprétation des données des capteurs

Logique de contrôle pour l'activation/désactivation de la pompe

Gestion des états des LED RGB selon l'état de santé de la plante

Implémentation d'un système d'alerte pour les conditions critiques

**Interface utilisateur :**

Développement d'une interface simple pour le contrôle manuel

Option de contrôle à distance via IR

**5. Plan de mise en œuvre**

**Phase 1 : Conception et achat :**

Sélection et achat des composants nécessaires

Conception initiale du schéma électrique et du montage

**Phase 2 : Assemblage et programmation :**

Assemblage du prototype

Programmation du microcontrôleur avec les fonctionnalités définies

**Phase 3 : Test :**

Test en laboratoire pour calibrer et ajuster les seuils des capteurs

Test de l'interface utilisateur et des fonctionnalités de contrôle

**Phase 4 : Installation et déploiement :**

Installation du système sur site

Surveillance et ajustements basés sur les observations en conditions réelles

**6. Critères de validation**

Le système doit être capable de surveiller avec précision les paramètres environnementaux.

Le système d'arrosage doit fonctionner automatiquement sans surveillance pendant au moins une semaine.

Les alertes et notifications doivent être clairement perceptibles et promptes.

L'interface utilisateur doit être intuitive et facile à utiliser.

**Mise En Œuvre**

**#### Introduction**

Ce projet IoT a été conçu pour surveiller et maintenir la santé des plantes en utilisant divers capteurs et actionneurs. Il offre des alertes sur plusieurs niveaux en fonction des conditions environnementales et permet un contrôle manuel ou à distance pour garantir un fonctionnement optimal. Les principales fonctionnalités incluent la détection de mouvement, la surveillance de la température et de l'humidité, le contrôle des niveaux d'eau, et la gestion de l'éclairage. Ce rapport détaillera les différents composants et fonctionnalités du système.

**#### Niveau 3 : Configuration Automatique et Alarme de Niveau Supérieur**

À ce niveau, le système réagit automatiquement aux conditions critiques :

- \*\*Détection de Mouvement et Température\*\* : Une lampe est allumée si un mouvement est détecté ou si la température descend en dessous de 5°C.

- \*\*Composants Utilisés\*\* : Capteur de mouvement (PIR), capteur de température et d'humidité (DHT11), relais pour le contrôle de la lampe.

**#### Niveau 2 : Alarme Sonore pour Conditions Critiques**

Ce niveau utilise un buzzer pour alerter en cas de conditions critiques :

- \*\*Température et Niveau d'Eau\*\* : Le buzzer s'active si la température dépasse 38°C ou descend en dessous de 0°C, ou si le niveau d'eau dans le réservoir est bas.

- \*\*Composants Utilisés\*\* : Capteur de température et d'humidité (DHT11), capteur de niveau d'eau, buzzer actif.

**#### Niveau 1 : Indication Visuelle de l'État de la Plante**

À ce niveau, un indicateur LED RGB informe sur l'état général de la plante en fonction de plusieurs facteurs :

- \*\*Humidité et Température\*\* : Les LEDs changent de couleur en fonction des seuils définis pour la température et l'humidité, indiquant visuellement l'état de la plante.

- \*\*Composants Utilisés\*\* : Capteur de température et d'humidité (DHT11), LEDs RGB.

**#### Indicateur de Luminosité**

Un indicateur LED jaune est utilisé pour donner des informations sur la luminosité ambiante :

- \*\*Luminosité\*\* : La LED jaune s'allume ou s'éteint en fonction de la lumière ambiante mesurée.

- \*\*Composants Utilisés\*\* : Capteur de lumière, LED jaune.

**#### Contrôle à Distance avec Télécommande IR**

Le système peut être contrôlé à distance via une télécommande infrarouge pour diverses actions :

- \*\*Fonctions de la Télécommande\*\* : Allumage/extinction du système, contrôle de la lampe et de la pompe.

- \*\*Composants Utilisés\*\* : Récepteur infrarouge, relais pour lampe et pompe.

**#### Contrôle Manuel avec Bouton Poussoir**

En plus du contrôle à distance, un bouton poussoir permet d'allumer ou d'éteindre le système manuellement :

- \*\*Fonctions du Bouton\*\* : Activation/désactivation du système en cas de panne de la commande à distance.

- \*\*Composants Utilisés\*\* : Bouton poussoir.

**### Conclusion**

Ce système IoT offre une solution complète pour la surveillance et l’entretien des plantes. Il intègre des capteurs de température, d'humidité et de mouvement, ainsi que des actionneurs tels que des relais et des LEDs. Les différentes alertes assurent une réponse adaptée aux conditions environnementales variées. De plus, le système permet un contrôle manuel et à distance, offrant une flexibilité maximale et une gestion efficace de la santé des plantes.

**Customized Code Genreralized**

**. Capteur d’humidité et temperature (DH11) :**

**#include <DHT.h>**

**#define DHTPIN 2 // Définir le pin de données du capteur DHT11**

**#define DHTTYPE DHT11 // Définir le type de capteur (DHT11 ou DHT22)**

**DHT dht(DHTPIN, DHTTYPE);**

**void setup() {**

**Serial.begin(9600);**

**dht.begin();**

**}**

**void loop() {**

**delay(2000); // Attente de 2 secondes entre les lectures**

**float humidity = dht.readHumidity(); // Lire l'humidité**

**float temperature = dht.readTemperature(); // Lire la température en Celsius**

**// Vérifier si la lecture a réussi**

**if (isnan(humidity) || isnan(temperature)) {**

**Serial.println("Échec de lecture du capteur DHT !");**

**return;**

**}**

**// Afficher les valeurs lues**

**Serial.print("Humidité: ");**

**Serial.print(humidity);**

**Serial.print("%\t");**

**Serial.print("Température: ");**

**Serial.print(temperature);**

**Serial.println("°C");**

**}**

**. Capteur de photorésistance :**

**#define PHOTO\_PIN 12 // Broche analogique utilisée pour le capteur de lumière**

**void setup() {**

**Serial.begin(9600);**

**}**

**void loop() {**

**// Lire la valeur du capteur de lumière**

**int lightValue = analogRead(PHOTO\_PIN);**

**// Afficher la valeur lue sur le moniteur série**

**Serial.print("Valeur du capteur de lumière : ");**

**Serial.println(lightValue);**

**// Attendre un court délai avant de lire à nouveau**

**delay(1000);}**

**. Capteur de niveau d’eau:**

**const int waterLevelPin = D4; // Broche à laquelle le capteur de niveau d'eau est connecté**

**void setup() {**

**Serial.begin(9600); // Démarre la communication série**

**pinMode(waterLevelPin, INPUT); // Définit la broche du capteur de niveau d'eau comme une entrée**

**}**

**void loop() {**

**int waterLevel = digitalRead(waterLevelPin); // Lecture de l'état du capteur de niveau d'eau**

**if (waterLevel == HIGH) {**

**Serial.println("Niveau d'eau élevé");**

**} else {**

**Serial.println("Niveau d'eau bas");**

**}**

**delay(1000); // Attente d'une seconde avant la prochaine lecture**

**}**

**. Capteur d’humidité au sol:**

**const int soilMoisturePin = A0; // Broche analogique à laquelle le capteur d'humidité du sol est connecté**

**void setup() {**

**Serial.begin(9600); // Démarre la communication série**

**}**

**void loop() {**

**int soilMoistureValue = analogRead(soilMoisturePin); // Lecture de la valeur analogique du capteur d'humidité du sol**

**// Convertit la valeur analogique en pourcentage d'humidité du sol**

**int moisturePercentage = map(soilMoistureValue, 0, 1023, 0, 100);**

**// Affichage de la valeur lue sur le moniteur série**

**Serial.print("Humidite du sol : ");**

**Serial.print(moisturePercentage);**

**Serial.println("%");**

**delay(1000); // Attente d'une seconde avant la prochaine lecture**

**}**

**. LED RGB:**

**// Broches pour la LED RGB (connectez-les aux broches PWM de votre ESP8266)**

**const int redPin = D1;**

**const int greenPin = D2;**

**const int bluePin = D3;**

**void setup() {**

**// Initialise les broches de la LED RGB en tant que sorties**

**pinMode(redPin, OUTPUT);**

**pinMode(greenPin, OUTPUT);**

**pinMode(bluePin, OUTPUT);**

**}**

**void loop() {**

**// Exemple de chaque fonction de base**

**// 1. Allumer la LED en rouge**

**setColor(255, 0, 0);**

**delay(1000);**

**// 2. Allumer la LED en vert**

**setColor(0, 255, 0);**

**delay(1000);**

**// 3. Allumer la LED en bleu**

**setColor(0, 0, 255);**

**delay(1000);**

**// 4. Mélanger les couleurs (cyan)**

**setColor(0, 255, 255);**

**delay(1000);**

**// 5. Mélanger les couleurs (magenta)**

**setColor(255, 0, 255);**

**delay(1000);**

**// 6. Mélanger les couleurs (jaune)**

**setColor(255, 255, 0);**

**delay(1000);**

**// 7. Éteindre la LED**

**turnOff();**

**delay(1000);**

**}**

**// Fonction pour définir la couleur de la LED RGB**

**void setColor(int redValue, int greenValue, int blueValue) {**

**analogWrite(redPin, redValue); // Définit l'intensité lumineuse de la LED rouge**

**analogWrite(greenPin, greenValue); // Définit l'intensité lumineuse de la LED verte**

**analogWrite(bluePin, blueValue); // Définit l'intensité lumineuse de la LED bleue**

**}**

**// Fonction pour éteindre la LED RGB**

**void turnOff() {**

**setColor(0, 0, 0); // Définit toutes les couleurs à 0 (éteint la LED)**

**}**

**#include <FastLED.h>**

**// Configuration de la LED RGB**

**#define NUM\_LEDS 1**

**#define DATA\_PIN D1**

**// Déclaration de la structure des LEDs**

**CRGB leds[NUM\_LEDS];**

**void setup() {**

**FastLED.addLeds<WS2812, DATA\_PIN, GRB>(leds, NUM\_LEDS);**

**}**

**void loop() {**

**// Allume la LED en rouge**

**leds[0] = CRGB::Red;**

**FastLED.show();**

**delay(1000);**

**// Allume la LED en vert**

**leds[0] = CRGB::Green;**

**FastLED.show();**

**delay(1000);**

**// Allume la LED en bleu**

**leds[0] = CRGB::Blue;**

**FastLED.show();**

**delay(1000);**

**// Allume la LED en blanc**

**leds[0] = CRGB::White;**

**FastLED.show();**

**delay(1000);**

**// Éteint la LED**

**leds[0] = CRGB::Black;**

**FastLED.show();**

**delay(1000);**

**}**

**// Définit une couleur personnalisée (rouge vif)**

**leds[0] = CRGB(255, 0, 0);**

**// Animation de dégradé de couleur**

**for (int i = 0; i < 256; i++) {**

**leds[0] = CHSV(i, 255, 255); // Variation de la teinte**

**FastLED.show();**

**delay(10);**

**}**

**// Définit une couleur avec une luminosité réduite**

**leds[0] = CRGB(255, 0, 0);**

**leds[0].fadeLightBy(128); // Réduit la luminosité de moitié**

**. Buzzer Actif:**

**// Broche pour le buzzer actif**

**const int buzzerPin = D1;**

**void setup() {**

**// Initialise la broche du buzzer actif en tant que sortie**

**pinMode(buzzerPin, OUTPUT);**

**}**

**void loop() {**

**// Exemple de chaque fonction de base**

**// 1. Activer le buzzer pour une durée de 1 seconde**

**buzz(1000);**

**delay(1000);**

**// 2. Activer le buzzer pour une durée de 500 millisecondes**

**buzz(500);**

**delay(1000);**

**// 3. Activer le buzzer pour une durée de 2 secondes**

**buzz(2000);**

**delay(1000);**

**// 4. Activer le buzzer pour une durée de 300 millisecondes**

**buzz(300);**

**delay(1000);**

**// 5. Activer le buzzer pour une durée de 1.5 secondes**

**buzz(1500);**

**delay(1000);**

**// 6. Activer le buzzer pour une durée de 400 millisecondes**

**buzz(400);**

**delay(1000);**

**// 7. Désactiver le buzzer**

**noBuzz();**

**delay(1000);**

**}**

**// Fonction pour activer le buzzer pendant une durée spécifiée en millisecondes**

**void buzz(unsigned int duration) {**

**tone(buzzerPin, 1000); // Joue un signal à une fréquence de 1000 Hz sur la broche du buzzer**

**delay(duration); // Active le buzzer pendant la durée spécifiée**

**noBuzz(); // Désactive le buzzer après la durée spécifiée**

**}**

**// Fonction pour désactiver le buzzer**

**void noBuzz() {**

**noTone(buzzerPin); // Arrête de jouer le signal sur la broche du buzzer**

**}**

**#include <ToneAC.h>**

**// Broche pour le buzzer actif**

**const int buzzerPin = D1;**

**void setup() {**

**// Initialise la broche du buzzer actif**

**pinMode(buzzerPin, OUTPUT);**

**}**

**void loop() {**

**// Joue une note C4 pendant 1000 ms**

**toneAC(buzzerPin, NOTE\_C4, 1000);**

**delay(1000);**

**}**

**void loop() {**

**// Joue la note Do (C4) pendant 500 ms**

**tone(buzzerPin, 261.63); // Fréquence en Hz pour la note C4**

**delay(500);**

**noTone(buzzerPin);**

**delay(500);}**

**void loop() {**

**// Active le buzzer à 50% de sa puissance pendant 1 seconde**

**analogWrite(buzzerPin, 128);**

**delay(1000);**

**analogWrite(buzzerPin, 0); // Désactive le buzzer**

**delay(1000);**

**}**

**void loop() {**

**// Tableau de fréquences des notes**

**int notes[] = {262, 294, 330, 349, 392, 440, 494, 523};**

**// Tableau de durées des notes (en millisecondes)**

**int durations[] = {500, 500, 500, 500, 500, 500, 500, 500};**

**// Joue la mélodie**

**for (int i = 0; i < 8; i++) {**

**tone(buzzerPin, notes[i], durations[i]);**

**delay(durations[i] + 50); // Ajoute un petit délai entre les notes**

**}**

**}**

**. Relais Avec une Pompe à eau :**

**Le code de base que j'ai fourni pour contrôler un relais avec une carte ESP32 est suffisant pour activer et désactiver le relais, ce qui permet de couper ou d'alimenter l'appareil électrique connecté au relais, comme votre pompe à eau.**

**Ce code permet d'allumer ou d'éteindre la pompe à eau en activant ou désactivant le relais, respectivement. Cela vous donne un contrôle de base sur l'alimentation de la pompe à eau à partir de votre ESP32.**

**Cependant, pour une application pratique, vous pourriez avoir besoin d'ajouter des fonctionnalités supplémentaires, telles que :**

**Gestion des erreurs : Vérification de l'état du relais pour s'assurer qu'il est activé ou désactivé avec succès.**

**Gestion du temps : Programmation de routines pour activer ou désactiver la pompe à eau à des heures spécifiques.**

**Gestion de l'état : Garder une trace de l'état actuel de la pompe à eau (allumée ou éteinte).**

**Sécurité : Ajout de mécanismes de sécurité pour détecter les conditions anormales, telles que la surchauffe ou la surcharge de la pompe à eau.**

**Interface utilisateur : Intégration d'une interface utilisateur pour permettre à l'utilisateur de contrôler manuellement la pompe à eau.**

**// Broche pour le relais**

**const int relayPin = D1;**

**void setup() {**

**// Initialise la broche du relais en tant que sortie**

**pinMode(relayPin, OUTPUT);**

**}**

**void loop() {**

**// Exemple de chaque fonction de base**

**// 1. Activer le relais (fermer le circuit)**

**turnOnRelay();**

**delay(1000);**

**// 2. Désactiver le relais (ouvrir le circuit)**

**turnOffRelay();**

**delay(1000);**

**}**

**// Fonction pour activer le relais (fermer le circuit)**

**void turnOnRelay() {**

**digitalWrite(relayPin, HIGH);**

**}**

**// Fonction pour désactiver le relais (ouvrir le circuit)**

**void turnOffRelay() {**

**digitalWrite(relayPin, LOW);**

**}**

**. Relais Avec une Pompe à eau :**

**Récepteur IR : Connectez la broche de signal du récepteur IR à une broche numérique de votre ESP32 (par exemple, D2). Assurez-vous également de connecter le récepteur IR à la masse (GND) et à une alimentation de 3.3V ou 5V.**

**Émetteur IR : Connectez la broche de signal de l'émetteur IR à une broche numérique de votre ESP32 (par exemple, D3). Assurez-vous également de connecter l'émetteur IR à la masse (GND) et à une alimentation de 3.3V ou 5V.**

**Programmation (exemple de code de base) :**

**Vous aurez besoin d'utiliser une bibliothèque pour le récepteur IR, comme la bibliothèque IRremoteESP8266 pour ESP8266 ou la bibliothèque IRremote pour Arduino. Pour l'émetteur IR, vous pouvez utiliser la même bibliothèque ou des bibliothèques similaires.**

Exemple de code pour l'émetteur IR :

**#include <IRremoteESP8266.h>**

**#include <IRrecv.h>**

**#include <IRutils.h>**

**const uint16\_t kRecvPin = D2; // Broche de réception IR**

**IRrecv irrecv(kRecvPin);**

**decode\_results results;**

**void setup() {**

**Serial.begin(115200);**

**irrecv.enableIRIn(); // Démarre la réception IR**

**}**

**void loop() {**

**if (irrecv.decode(&results)) {**

**// Affiche le code de la télécommande**

**Serial.println(results.value, HEX);**

**irrecv.resume(); // Continue la réception IR**

**}**

**}**

Exemple de code pour le récépteur IR :

**#include <IRremoteESP8266.h>**

**#include <IRsend.h>**

**#include <IRutils.h>**

**const uint16\_t kIrLedPin = D3; // Broche de l'émetteur IR**

**IRsend irsend(kIrLedPin);**

**void setup() {**

**Serial.begin(115200);**

**}**

**void loop() {**

**irsend.sendNEC(0x00FF30CF, 32); // Code NEC pour la commande ON/OFF**

**delay(1000); // Attente entre les transmissions**

**}**

**Fonctionnalités supplémentaires et bibliothèques :**

**Analyse des codes IR : Utilisez la fonction decode() pour décoder les signaux IR reçus et identifier les commandes de la télécommande.**

**Transmission de commandes IR : Utilisez différentes fonctions de la bibliothèque IR pour envoyer des signaux IR avec des protocoles comme NEC, Sony, etc.**

**Gestion des commandes : Créez une correspondance entre les codes IR reçus et les actions à effectuer dans votre système de santé.**

**. Boutton Pousseoir :**

**Connexion physique :**

**Connectez une broche du bouton-poussoir à une broche numérique de votre ESP32 (par exemple, D4). Connectez l'autre broche du bouton-poussoir à la masse (GND) de votre ESP32.**

Programmation (exemple de code de base) :

**Voici un exemple de code qui utilise un bouton-poussoir pour contrôler l'état de votre système (allumé ou éteint) :gérez des commandes IR dans une interface utilisateur pour permettre aux utilisateurs de contrôler votre système à distance :**

**const int buttonPin = D4; // Broche à laquelle le bouton est connecté**

**bool systemState = false; // État initial du système (éteint)**

**void setup() {**

**pinMode(buttonPin, INPUT\_PULLUP); // Configure la broche du bouton comme entrée avec une résistance de tirage**

**Serial.begin(115200);**

**}**

**void loop() {**

**// Lecture de l'état du bouton**

**int buttonState = digitalRead(buttonPin);**

**// Si le bouton est enfoncé (état bas), bascule l'état du système**

**if (buttonState == LOW) {**

**systemState = !systemState; // Bascule l'état du système**

**delay(100); // Anti-rebond (débouncing)**

**}**

**// Si le système est allumé, effectuez les actions correspondantes**

**if (systemState) {**

**// Actions lorsque le système est allumé**

**Serial.println("Le système est allumé !");**

**} else {**

**// Actions lorsque le système est éteint**

**Serial.println("Le système est éteint !");**

**}**

**}**

Fonctionnalités supplémentaires et bibliothèques :

**Anti-rebond (debouncing) : Lorsqu'un bouton est enfoncé, il peut rebondir entre l'état haut et l'état bas pendant une très courte période. Vous pouvez ajouter un délai après la détection d'un changement d'état pour éviter les faux déclenchements.**

**Gestion de l'état : Utilisez la variable systemState pour suivre l'état actuel du système et effectuer des actions en fonction de cet état.**

**Interface utilisateur : Intégrez des actions spécifiques à l'état du système dans votre code, comme allumer ou éteindre des capteurs ou des dispositifs, en fonction de l'état du bouton.**