TP8

L'objectif de ce code est de créer un système de surveillance en temps réel de la température, de l'humidité de l'air (via un capteur DHT11) et de l'humidité du sol (via un capteur d'humidité du sol) connecté à un NodeMCU ESP8266. Le NodeMCU agit comme un serveur web qui héberge une interface utilisateur accessible via un navigateur. Cette interface affiche les données de température, d'humidité de l'air, d'humidité du sol, ainsi que l'heure et la date actuelles, mises à jour en temps réel.

```
//----------------------------------------Include the NodeMCU ESP8266 Library
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include "DHT.h"  // Include the DHT Sensor Library
#define DHTTYPE DHT11 // Defines the type of DHT sensor used (DHT11)
#include "SITD-DHT-Soil.h" // Include the contents of the User Interface Web page
#define LEDonBoard 2  // On Board LED for connection indicators
#define SOIL_MOISTURE_PIN A0 // Pin for soil moisture sensor (A0)
//----------------------------------------SSID and Password of your WiFi router
const char *ssid = "XXXXXX";
const char *password = "XXXXXXXX";
ESP8266WebServer server(80);  // Server on port 80
const int DHTPin = 5; // Pin used for the DHT11 sensor (D1 = GPIO 5)
DHT dht(DHTPin, DHTTYPE); // Initialize DHT sensor
//----------------------------------------This routine is executed when you open NodeMCU ESP8266 IP Address in browser
void handleRoot() {
String s = MAIN_page; // Read HTML contents
server.send(200, "text/html", s); // Send web page
}
//----------------------------------------Procedure for reading the temperature value of a DHT11 sensor
void handleDHT11Temperature() {
float t = dht.readTemperature();  // Read temperature as Celsius
String Temperature_Value = String(t);
server.send(200, "text/plane", Temperature_Value); // Send Temperature value to client
if (isnan(t)) {
Serial.println("Failed to read from DHT sensor!");
} else {
Serial.print("DHT11 || Temperature : ");
Serial.print(t);
Serial.print(" || ");
}
}
```

```
//---------------------------------------Procedure for reading humidity values from DHT11 sensors
void handleDHT11Humidity() {
float h = dht.readHumidity(); // Read humidity
String Humidity_Value = String(h);
server.send(200, "text/plane", Humidity_Value); // Send Humidity value to client
if (isnan(h)) {
Serial.println("Failed to read from DHT sensor!");
} else {
Serial.print("Humidity : ");
Serial.println(h);
}
}
//---------------------------------------Procedure for reading soil moisture values
void handleSoilMoisture() {
int soilMoistureValue = analogRead(SOIL_MOISTURE_PIN); // Read soil moisture value
String SoilMoisture_Value = String(soilMoistureValue);
server.send(200, "text/plane", SoilMoisture_Value); // Send Soil Moisture value to client
Serial.print("Soil Moisture : ");
Serial.println(soilMoistureValue);
}
//---------------------------------------Setup
void setup(void) {
Serial.begin(115200);
delay(500);
dht.begin();  // Start reading DHT11 sensors
pinMode(SOIL_MOISTURE_PIN, INPUT); // Set soil moisture pin as input
delay(500);
WiFi.begin(ssid, password); // Connect to WiFi
Serial.println("");
pinMode(LEDonBoard, OUTPUT); // On Board LED port Direction output
digitalWrite(LEDonBoard, HIGH); // Turn off LED On Board
// Wait for connection
Serial.print("Connecting");
while (WiFi.status() != WL_CONNECTED) {
Serial.print(".");
digitalWrite(LEDonBoard, LOW);
delay(250);
digitalWrite(LEDonBoard, HIGH);
delay(250);
}
```

```
digitalWrite(LEDonBoard, HIGH); // Turn off LED when connected
Serial.println("");
Serial.print("Successfully connected to : ");
Serial.println(ssid);
Serial.print("IP address: ");
Serial.println(WiFi.localIP());
// Configure server routes
server.on("/", handleRoot); // Handle root location
server.on("/readTemperature", handleDHT11Temperature); // Handle temperature requests
server.on("/readHumidity", handleDHT11Humidity); // Handle humidity requests
server.on("/readSoilMoisture", handleSoilMoisture); // Handle soil moisture requests
server.begin(); // Start server
Serial.println("HTTP server started");
}
//---------------------------------------Loop
void loop() {
server.handleClient();  // Handle client requests
}
```

User Interface Web page (SITD-DHT-Soil)

```
const char MAIN_page[] PROGMEM = R"=====(
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link href="https://use.fontawesome.com/releases/v5.7.2/css/all.css" rel="stylesheet">
<style>
html {
font-family: Arial;
display: inline-block;
margin: 0px auto;
text-align: center;
}
h1 { font-size: 2.0rem; }
p { font-size: 2.0rem; }
.units { font-size: 1.2rem; }
.dht-labels {
font-size: 1.5rem;
vertical-align: middle;
padding-bottom: 15px;
}
</style>
</head>
<body>
<h1>NodeMCU ESP8266 Monitoring Sensors</h1>
<p>
<i class="fa fa-thermometer-half" style="font-size:3.0rem;color:#62a1d3;"></i>
<span class="dht-labels">Temperature : </span>
<span id="TemperatureValue">0</span>
<sup class="units">&deg;C</sup>
</p>
<p>
<i class="fa fa-tint" style="font-size:3.0rem;color:#75e095;"></i>
<span class="dht-labels">Humidity : </span>
<span id="HumidityValue">0</span>
<sup class="units">%</sup>
</p>
<p>
<i class="fas fa-seedling" style="font-size:3.0rem;color:#8e44ad;"></i>
```

```
<span class="dht-labels">Soil Moisture : </span>
<span id="SoilMoistureValue">0</span>
<sup class="units">(Analog)</sup>
</p>
<p>
<i class="far fa-clock" style="font-size:1.0rem;color:#e3a8c7;"></i>
<span style="font-size:1.0rem;">Time </span>
<span id="time" style="font-size:1.0rem;"></span>
<i class="far fa-calendar-alt" style="font-size:1.0rem;color:#f7dc68";></i>
<span style="font-size:1.0rem;">Date </span>
<span id="date" style="font-size:1.0rem;"></span>
</p>
<script>
setInterval(function() {
// Call functions repeatedly with 2-second intervals
getTemperatureData();
getHumidityData();
getSoilMoistureData();
}, 2000);
setInterval(function() {
// Update time and date every second
Time_Date();
}, 1000);
function getTemperatureData() {
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
if (this.readyState == 4 && this.status == 200) {
document.getElementById("TemperatureValue").innerHTML = this.responseText;
}
};
xhttp.open("GET", "readTemperature", true);
xhttp.send();
}
function getHumidityData() {
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
if (this.readyState == 4 && this.status == 200) {
document.getElementById("HumidityValue").innerHTML = this.responseText;
}
};
```

```
xhttp.open("GET", "readHumidity", true);
xhttp.send();
}
function getSoilMoistureData() {
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
if (this.readyState == 4 && this.status == 200) {
document.getElementById("SoilMoistureValue").innerHTML = this.responseText;
}
};
xhttp.open("GET", "readSoilMoisture", true);
xhttp.send();
}
function Time_Date() {
var t = new Date();
document.getElementById("time").innerHTML = t.toLocaleTimeString();
var d = new Date();
const dayNames = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday","Saturday"];
const monthNames = ["January", "February", "March", "April", "May", "June","July", "August",
"September", "October", "November", "December"];
document.getElementById("date").innerHTML = dayNames[d.getDay()] + ", " + d.getDate() + "-" +
monthNames[d.getMonth()] + "-" + d.getFullYear();
}
</script>
</body>
</html>
)=====";
```

Explications

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include "DHT.h"
```

ESP8266WiFi.h : Bibliothèque pour gérer la connexion WiFi du NodeMCU.

WiFiClient.h : Bibliothèque pour gérer les connexions clientes pour le serveur web.

ESP8266WebServer.h : Bibliothèque pour créer un serveur web sur le NodeMCU.

DHT.h : Bibliothèque pour interagir avec le capteur DHT11.

```
#define DHTTYPE DHT11 // Type de capteur DHT utilisé (DHT11)
#define LEDonBoard 2  // Broche de la LED intégrée du NodeMCU (GPIO 2)
#define SOIL_MOISTURE_PIN A0 // Broche analogique pour le capteur d'humidité du sol
```

DHTTYPE DHT11 : Définit le type de capteur DHT utilisé (ici, un DHT11).

LEDonBoard : Broche de la LED intégrée du NodeMCU, utilisée comme indicateur visuel.

SOIL_MOISTURE_PIN : Broche analogique (A0) pour lire les données du capteur d'humidité du sol.

```
const char *ssid = "SITD"; // Nom du réseau WiFi
const char *password = "XXXXXXXX"; // Mot de passe du réseau WiFi
```

ssid : Nom du réseau WiFi auquel le NodeMCU se connecte.

password : Mot de passe du réseau WiFi.

```
ESP8266WebServer server(80);  // Serveur web sur le port 80
const int DHTPin = 5; // Broche numérique pour le capteur DHT11 (D1 = GPIO 5)
DHT dht(DHTPin, DHTTYPE); // Initialisation du capteur DHT
```

server(80) : Crée un serveur web qui écoute sur le port 80 (port standard pour HTTP).

DHTPin : Broche numérique (D1) pour le capteur DHT11.

dht(DHTPin, DHTTYPE) : Initialise le capteur DHT avec la broche et le type définis.

```
int readsuccess; // Variable pour vérifier si la lecture de l'UID est réussie
byte readcard[4]; // Tableau pour stocker l'UID de la carte (4 octets)
char str[32] = ""; // Chaîne de caractères pour stocker l'UID au format texte
String StrUID; // Chaîne de caractères pour stocker l'UID au format String
```

readsuccess : Indique si la lecture de l'UID a réussi (1) ou échoué (0).

readcard : Stocke l'UID de la carte sous forme de tableau de 4 octets.

str : Stocke l'UID converti en chaîne de caractères hexadécimale.

StrUID : Stocke l'UID au format String pour un affichage facile.

```
void handleRoot() {
  String s = MAIN_page; // Lit le contenu HTML de la page web
  server.send(200, "text/html", s); // Envoie la page web au client
}
```

handleRoot() : Cette fonction est appelée lorsque l'utilisateur accède à la page racine (/) du serveur web. Elle envoie la page HTML stockée dans MAIN_page au client.

```
void handleDHT11Temperature() {
  float t = dht.readTemperature();  // Lit la température en Celsius
```

```
  String Temperature_Value = String(t);
  server.send(200, "text/plane", Temperature_Value); // Envoie la température au client
  if (isnan(t)) {
   Serial.println("Failed to read from DHT sensor!");
  } else {
   Serial.print("DHT11 || Temperature : ");
   Serial.print(t);
   Serial.print(" || ");
  }
 }
```

handleDHT11Temperature() : Lit la température à partir du capteur DHT11 et l'envoie au client via une requête HTTP.

isnan(t) : Vérifie si la lecture de la température a échoué.

```
 void handleDHT11Humidity() {
  float h = dht.readHumidity(); // Lit l'humidité de l'air
  String Humidity_Value = String(h);
  server.send(200, "text/plane", Humidity_Value); // Envoie l'humidité au client
  if (isnan(h)) {
   Serial.println("Failed to read from DHT sensor!");
  } else {
   Serial.print("Humidity : ");
   Serial.println(h);
  }
 }
```

handleDHT11Humidity() : Lit l'humidité de l'air à partir du capteur DHT11 et l'envoie au client via une requête HTTP.

```
 void handleSoilMoisture() {
  int soilMoistureValue = analogRead(SOIL_MOISTURE_PIN); // Lit la valeur du capteur d'humidité du sol
  String SoilMoisture_Value = String(soilMoistureValue);
  server.send(200, "text/plane", SoilMoisture_Value); // Envoie la valeur au client
  Serial.print("Soil Moisture : ");
  Serial.println(soilMoistureValue);
 }
```

handleSoilMoisture() : Lit la valeur analogique du capteur d'humidité du sol et l'envoie au client via une requête HTTP.

Fonction setup()

setup() : Initialise les composants, configure la connexion WiFi, et démarre le serveur web.

```
 void loop() {
  server.handleClient();  // Gère les requêtes des clients
 }
```

loop() : Cette fonction est exécutée en boucle pour gérer les requêtes des clients connectés au serveur web.

User Interface Web page (SITD-DHT-Soil)

En-tête (<head>) :

<meta name="viewport" content="width=device-width, initial-scale=1"> : Rend la page web responsive pour les appareils mobiles.

<link href="https://use.fontawesome.com/releases/v5.7.2/css/all.css" rel="stylesheet"> : Inclut les icônes de FontAwesome pour une meilleure présentation.

<style> : Définit les styles CSS pour la mise en page, les polices, les tailles de texte, etc.

html : Applique une police Arial, centre le texte et affiche les éléments en bloc.

h1 : Définit la taille de la police pour le titre principal.

p : Définit la taille de la police pour les paragraphes.

.units : Définit la taille de la police pour les unités de mesure.

.dht-labels : Définit la taille de la police et l'alignement vertical pour les étiquettes de température, d'humidité et d'humidité du sol.

Corps (<body>) :

<h1> : Affiche le titre principal de la page : "NodeMCU ESP8266 Monitoring Sensors".

<p> : Affiche les valeurs de température, d'humidité de l'air et d'humidité du sol avec des icônes.

<i class="fa fa-thermometer-half"> : Icône pour la température.

 : Emplacement où la température sera affichée dynamiquement.

<i class="fa fa-tint"> : Icône pour l'humidité de l'air.

 : Emplacement où l'humidité de l'air sera affichée dynamiquement.

<i class="fas fa-seedling"> : Icône pour l'humidité du sol.

 : Emplacement où l'humidité du sol sera affichée dynamiquement.

<sup class="units"> : Affiche les unités de mesure (°C, %, Analog).

<p> : Affiche l'heure et la date actuelles.

<i class="far fa-clock"> : Icône pour l'heure.

 : Emplacement où l'heure sera affichée dynamiquement.

<i class="far fa-calendar-alt"> : Icône pour la date.

 : Emplacement où la date sera affichée dynamiquement.

Script JavaScript :

setInterval(function() { ... }, 2000); : Appelle les fonctions getTemperatureData(), getHumidityData() et getSoilMoistureData() toutes les 2 secondes pour mettre à jour les valeurs de température, d'humidité de l'air et d'humidité du sol.

setInterval(function() { ... }, 1000); : Appelle la fonction Time_Date() toutes les secondes pour mettre à jour l'heure et la date.

getTemperatureData() : Envoie une requête AJAX au serveur pour obtenir la dernière valeur de température et met à jour l'élément HTML correspondant.

xhttp.onreadystatechange : Définit une fonction de rappel qui est appelée lorsque l'état de la requête change.

xhttp.open("GET", "readTemperature", true) : Ouvre une requête GET vers l'endpoint /readTemperature du serveur.

xhttp.send() : Envoie la requête au serveur.

getHumidityData() : Envoie une requête AJAX au serveur pour obtenir la dernière valeur d'humidité de l'air et met à jour l'élément HTML correspondant.

getSoilMoistureData() : Envoie une requête AJAX au serveur pour obtenir la dernière valeur d'humidité du sol et met à jour l'élément HTML correspondant.

Time_Date() : Met à jour l'heure et la date actuelles en utilisant l'objet Date de JavaScript.

t.toLocaleTimeString() : Convertit l'heure actuelle en une chaîne de caractères lisible.

dayNames[d.getDay()] : Obtient le nom du jour de la semaine.

monthNames[d.getMonth()] : Obtient le nom du mois.

TP9

L'objectif de ce code est de lire les données de température et d'humidité à partir d'un capteur DHT11 connecté à un NodeMCU ESP8266, puis d'envoyer ces données à un serveur ThingSpeak via une connexion WiFi. ThingSpeak est une plateforme IoT qui permet de stocker, visualiser et analyser des données en temps réel.

```
//----------------------------------------Include the NodeMCU ESP8266 Library
#include <ESP8266WiFi.h>
//----------------------------------------
#include "DHT.h"  //--> Include the DHT Sensor Library
#define DHTTYPE DHT11 //--> Defines the type of DHT sensor used (DHT11, DHT21, and DHT22), in this project the sensor used is DHT11.
#define LEDonBoard 2  //--> Defining an On Board LED, used for indicators when the process of connecting to a wifi router
String apiKey = "ThingSpeak_API_Key"; //--> Enter your Write API key from ThingSpeak
//----------------------------------------SSID and Password of your WiFi router
const char* ssid = "xxxxxx";
const char* password = "xxxxxxxx";
//----------------------------------------ThingSpeak Server
const char* server = "api.thingspeak.com";
//----------------------------------------
const int DHTPin = 5; //--> The pin used for the DHT11 sensor is Pin D1=Pin 5
DHT dht(DHTPin, DHTTYPE); //--> Initialize DHT sensor, DHT dht(Pin_used, Type_of_DHT_Sensor);
WiFiClient client;
void setup() {
// put your setup code here, to run once:
Serial.begin(115200);
delay(500);
dht.begin();  //--> Start reading DHT11 sensors
delay(500);
WiFi.begin(ssid, password); //--> Connect to your WiFi router
Serial.println("");
pinMode(LEDonBoard,OUTPUT); //--> On Board LED port Direction output
digitalWrite(LEDonBoard, HIGH); //--> Turn off Led On Board
//----------------------------------------Wait for connection
Serial.print("Connecting");
while (WiFi.status() != WL_CONNECTED) {
Serial.print(".");
//----------Make the On Board Flashing LED on the process of connecting to the wifi router.
digitalWrite(LEDonBoard, LOW);
delay(250);
```

```
digitalWrite(LEDonBoard, HIGH);
delay(250);
}
digitalWrite(LEDonBoard, HIGH);
//-------------------------If connection successful show IP address in serial monitor
Serial.println("");
Serial.print("Successfully connected to : ");
Serial.println(ssid);
}
void loop() {
// put your main code here, to run repeatedly:
float h = dht.readHumidity();
float t = dht.readTemperature();
if (isnan(h) || isnan(t)) {
Serial.println("Failed to read from DHT sensor!");
return;
}
if (client.connect(server,80))  {   //--> "184.106.153.149" or api.thingspeak.com
String postStr = apiKey;
postStr +="&field1=";
postStr += String(t);
postStr +="&field2=";
postStr += String(h);
postStr += "\r\n\r\n";
client.print("POST /update HTTP/1.1\n");
client.print("Host: api.thingspeak.com\n");
client.print("Connection: close\n");
client.print("X-THINGSPEAKAPIKEY: "+apiKey+"\n");
client.print("Content-Type: application/x-www-form-urlencoded\n");
client.print("Content-Length: ");
client.print(postStr.length());
client.print("\n\n");
client.print(postStr);
Serial.print("Temperature: ");
Serial.print(t);
Serial.print(" degrees Celcius, Humidity: ");
Serial.print(h);
Serial.println("%. Send to Thingspeak.");
}
client.stop();
```

```
Serial.println("Waiting...");
// thingspeak needs minimum 15 sec delay between updates
digitalWrite(LEDonBoard, LOW);
delay(250);
digitalWrite(LEDonBoard, HIGH);
delay(750);
}
```

Explications

```
#include <ESP8266WiFi.h>
#include "DHT.h"
```

ESP8266WiFi.h : Bibliothèque pour gérer la connexion WiFi du NodeMCU.

DHT.h : Bibliothèque pour interagir avec le capteur DHT11.

```
#define DHTTYPE DHT11 // Type de capteur DHT utilisé (DHT11)
#define LEDonBoard 2  // Broche de la LED intégrée du NodeMCU (GPIO 2)
String apiKey = "ThingSpeak_API_Key"; // Clé API de ThingSpeak pour envoyer des données
```

DHTTYPE DHT11 : Définit le type de capteur DHT utilisé (ici, un DHT11).

LEDonBoard : Broche de la LED intégrée du NodeMCU, utilisée comme indicateur visuel.

apiKey : Clé API de ThingSpeak pour autoriser l'envoi de données.

```
const char* ssid = "xxxxxx"; // Nom du réseau WiFi
const char* password = "xxxxxxxx"; // Mot de passe du réseau WiFi
```

ssid : Nom du réseau WiFi auquel le NodeMCU se connecte.

password : Mot de passe du réseau WiFi.

```
const char* server = "api.thingspeak.com"; // Adresse du serveur ThingSpeak
```

server : Adresse du serveur ThingSpeak où les données seront envoyées.

```
const int DHTPin = 5; // Broche numérique pour le capteur DHT11 (D1 = GPIO 5)
DHT dht(DHTPin, DHTTYPE); // Initialisation du capteur DHT
WiFiClient client; // Objet pour gérer la connexion WiFi
```

DHTPin : Broche numérique (D1) pour le capteur DHT11.

dht(DHTPin, DHTTYPE) : Initialise le capteur DHT avec la broche et le type définis.

client : Objet pour gérer la connexion WiFi et envoyer des données à ThingSpeak.

Fonction setup()

Serial.begin(115200) : Initialise la communication série pour le débogage.

dht.begin() : Initialise le capteur DHT.

WiFi.begin(ssid, password) : Se connecte au réseau WiFi.

pinMode(LEDonBoard, OUTPUT) : Configure la LED intégrée comme sortie.

digitalWrite(LEDonBoard, HIGH) : Éteint la LED.

while (WiFi.status() != WL_CONNECTED) : Attend que la connexion WiFi soit établie tout en faisant clignoter la LED.

Fonction loop()

Lecture des données :

float h = dht.readHumidity(); : Lit l'humidité de l'air.

float t = dht.readTemperature(); : Lit la température.

Vérification des données :

if (isnan(h) || isnan(t)) : Vérifie si la lecture des données a échoué. Si c'est le cas, affiche un message d'erreur et quitte la fonction.

Connexion au serveur ThingSpeak :

if (client.connect(server, 80)) : Se connecte au serveur ThingSpeak sur le port 80 (HTTP).

Préparation des données :

String postStr = apiKey; : Commence à construire la chaîne de données à envoyer.

postStr += "&field1="; : Ajoute le champ pour la température.

postStr += String(t); : Ajoute la valeur de la température.

postStr += "&field2="; : Ajoute le champ pour l'humidité.

postStr += String(h); : Ajoute la valeur de l'humidité.

Envoi des données :

client.print("POST /update HTTP/1.1\n"); : Envoie une requête HTTP POST à ThingSpeak.

client.print("Host: api.thingspeak.com\n"); : Spécifie l'hôte du serveur.

client.print("Connection: close\n"); : Ferme la connexion après l'envoi.

client.print("X-THINGSPEAKAPIKEY: " + apiKey + "\n"); : Ajoute la clé API pour l'autorisation.

client.print("Content-Type: application/x-www-form-urlencoded\n"); : Spécifie le type de contenu.

client.print("Content-Length: "); : Ajoute la longueur du contenu.

client.print(postStr.length()); : Ajoute la longueur de la chaîne de données.

client.print("\n\n"); : Ajoute une ligne vide pour terminer l'en-tête HTTP.

client.print(postStr); : Envoie les données.

Affichage des données dans le moniteur série :

Serial.print("Temperature: "); : Affiche la température dans le moniteur série.

Serial.print(t); : Affiche la valeur de la température.

Serial.print(" degrees Celcius, Humidity: "); : Affiche l'humidité dans le moniteur série.

Serial.print(h); : Affiche la valeur de l'humidité.

Serial.println("%. Send to Thingspeak."); : Affiche un message indiquant que les données ont été envoyées.

Fermeture de la connexion :

client.stop(); : Ferme la connexion avec le serveur ThingSpeak.

Délai entre les mises à jour :

delay(750); : Attend 750 ms avant de recommencer le processus.

digitalWrite(LEDonBoard, LOW); : Allume la LED intégrée.

delay(250); : Attend 250 ms.

digitalWrite(LEDonBoard, HIGH); : Éteint la LED intégré