Progetto IoT

Obiettivo del Progetto:

Realizzare un sistema di monitoraggio ambientale con Arduino MKR WiFi 1010 che legge i dati di temperatura e umidità da un sensore DHT11 e trasmette queste informazioni a un server remoto tramite MQTT. Il server, implementato utilizzando PHP, riceve i dati MQTT e li salva in un database. Inoltre, il sistema visualizza i dati in tempo reale su un display LCD e può attivare azioni come l'accensione di un LED, di una ventola o l'emissione di un suono tramite un buzzer in base ai valori rilevati.

Specifiche per il Database:

1. Nome del Database: AmbientMonitoringDB
2. Tabelle:
    - SensorData:
        - ID: Intero, chiave primaria auto-incrementale
        - Temperatura: Float, temperatura rilevata in gradi Celsius
        - Umidità: Float, umidità rilevata in percentuale
        - Timestamp: Data/ora della rilevazione, timestamp
3. Connessione al Database:
    - Utilizzare le funzionalità fornite da PHP per connettersi al database MySQL/MariaDB.
    - Memorizzare le credenziali di accesso al database in modo sicuro e non renderle pubbliche.

Implementazione del Database:

1. Creare il database e le tabelle utilizzando MySQL/MariaDB o un'altra piattaforma compatibile con PHP.
2. Scrivere uno script PHP che riceva i dati MQTT dal server MQTT e li salvi nel database.
3. Utilizzare le funzioni fornite dalla libreria `phpMQTT` per ricevere i dati MQTT e le funzioni PHP per interagire con il database.
4. Verificare che la connessione al database sia stabilita correttamente e che i dati vengano inseriti correttamente nella tabella `SensorData`.
5. Gestire eventuali errori di connessione al database o di inserimento dei dati.

Componenti Necessari:

- Arduino MKR WiFi 1010: Scheda Arduino con connettività WiFi integrata.
- Sensore DHT11: Per misurare temperatura e umidità.
- Display LCD: Per visualizzare i dati rilevati e altre informazioni.

- Ventola, Buzzer e LED: Per attuare azioni in base ai dati ambientali.
- Connessione WiFi: Accesso a una rete WiFi per comunicazione con il server remoto.
- Server Remoto: Un server (ad esempio, un computer) che riceve e gestisce i dati inviati dall'Arduino.
- Broker MQTT: Installazione di un broker MQTT sul server per la comunicazione MQTT.

Passaggi per il Progetto:

Configurazione Arduino:
Collega il sensore DHT11 e il display LCD all'Arduino MKR WiFi 1010.
Configura la connessione WiFi per collegarsi alla rete WiFi.
Installa la libreria PubSubClient per supportare la comunicazione MQTT.

Programmazione Arduino:
Scrivi un programma Arduino che legga i valori di temperatura e umidità dal sensore DHT11.
Configura e connetti Arduino al broker MQTT per inviare i dati al server.

Configurazione Server Remoto:
Installa e configura un broker MQTT (come Mosquitto) sul server remoto.
Implementa uno script PHP sul server per gestire la ricezione dei messaggi MQTT e il salvataggio dei dati in un database.

Gestione Dati sul Server:
Utilizza PHP per sottoscrivere e gestire i messaggi MQTT provenienti dall'Arduino.
Salva i dati di temperatura e umidità ricevuti dal sensore in un database (ad esempio, MySQL).

Interfaccia Utente e Azioni:
Aggiorna lo script PHP per elaborare i dati salvati nel database e attuare azioni in base ai valori rilevati (ad esempio, attivazione di LED, ventola o buzzer).

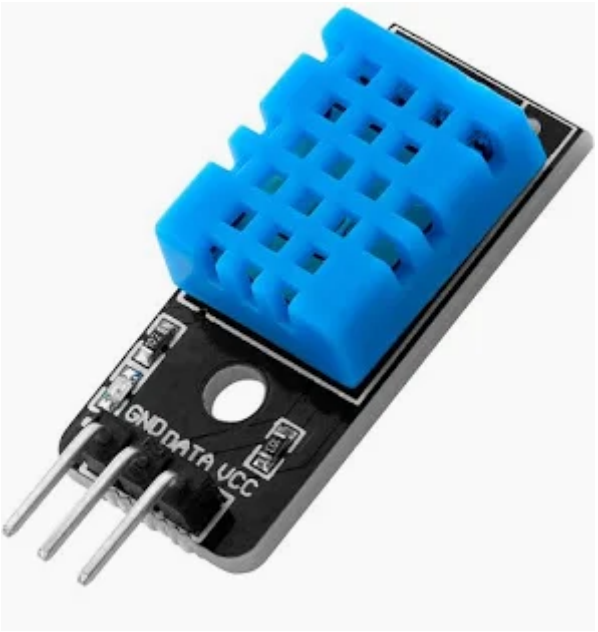
Visualizzazione dei Dati:
Utilizza il display LCD collegato all'Arduino per mostrare i dati di temperatura e umidità in tempo reale.
Implementa un'interfaccia utente web (opzionale) per visualizzare i dati storici e in tempo reale provenienti dal server.

Datasheet e Riferimenti:

1. Consulta i datasheet dei singoli componenti per informazioni dettagliate sui collegamenti e sull'utilizzo.
2. Segui gli esempi di codice forniti per implementare correttamente il funzionamento di ciascun componente.

Sensore DHT11 (Temperatura e Umidità):

Facile da usare per misurare la temperatura e l'umidità ambientale.



Collegamento del Sensore DHT11:

Pin DHT11:

- VCC: Collegare a 5V (o 3.3V se preferito)
- DATA: Collegare a un pin digitale di Arduino (ad esempio, pin 4)
- GND: Collegare a GND

Libreria e Script per Arduino:

Installazione della Libreria:

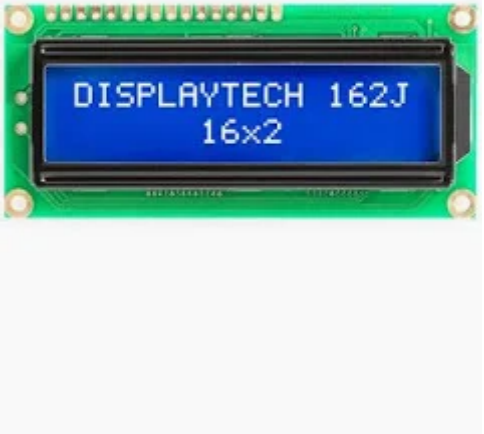
Apri l'IDE di Arduino e vai su "Sketch" -> "Include Library" -> "Manage Libraries".
Cerca e installa la libreria "DHT sensor library" di Adafruit.

Esempio di Lettura Dati:

```
#include <DHT.h>

#define DHTPIN 4
#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
  dht.begin();
}
```

```
void loop() {
  delay(2000);
  float temperature = dht.readTemperature();  // Temperatura in Celsius
  float humidity = dht.readHumidity();      // Leggi l'umidità relativa

  Serial.print("Temperatura: ");
  Serial.print(temperature);
  Serial.print(" °C, Umidità: ");
  Serial.print(humidity);
  Serial.println(" %");
}
```

Display LCD (16x2 o Simile):
Un display LCD 16x2 può essere utilizzato per visualizzare i dati rilevati dal sensore.



Collegamento del Display LCD:

Pin Display LCD:

- VCC: Collegare a 5V
- GND: Collegare a GND
- SDA: Collegare a un pin I2C SDA (Arduino MKR WiFi 1010 supporta la comunicazione I2C)
- SCL: Collegare a un pin I2C SCL

Libreria e Script per Arduino:

Installazione della Libreria:
Apri l'IDE di Arduino e vai su "Sketch" -> "Include Library" -> "Manage Libraries".
Cerca e installa la libreria "LiquidCrystal_I2C" di Frank de Brabander.

Esempio di Utilizzo:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 16, 2);  // Indirizzo I2C e dimensioni
(16x2)

void setup() {
  lcd.begin();  // Inizializza il display LCD
  lcd.backlight();  // Accendi la retroilluminazione

  lcd.print("Temperatura:");
}

void loop() {
  // Aggiorna la temperatura sul display LCD
  lcd.setCursor(0, 1);
  lcd.print("25.5 C");  // Esempio di temperatura da visualizzare
  delay(1000);  // Aggiorna ogni secondo
}
```

Ventola, Buzzer e LED:

Controllo di una Ventola (o Motore DC):

Collegamento della Ventola:

Collega il polo positivo della ventola a un pin digitale di Arduino.
Collega il polo negativo della ventola a GND.

Esempio di Attivazione della Ventola:

```
int ventolaPin = 6;  // Pin digitale per controllare la ventola
void setup() {
  pinMode(ventolaPin, OUTPUT);
}

void loop() {
  digitalWrite(ventolaPin, HIGH);  // Accendi la ventola
  delay(5000);  // Mantieni accesa la ventola per 5 secondi
  digitalWrite(ventolaPin, LOW);  // Spegni la ventola
  delay(5000);  // Attendi 5 secondi prima di ripetere
}
```

Controllo di un Buzzer:

Collegamento del Buzzer:
Collega uno dei pin del buzzer a un pin digitale di Arduino.
Collega l'altro pin del buzzer a GND.

Esempio di Emissione di Suono con il Buzzer:

```
int buzzerPin = 7;  // Pin digitale per controllare il buzzer

void setup() {
  pinMode(buzzerPin, OUTPUT);
}

void loop() {
  tone(buzzerPin, 1000);  // Emetti un suono a 1000Hz
  delay(1000);  // Emetti il suono per 1 secondo
  noTone(buzzerPin);  // Smetti di emettere il suono
  delay(1000);  // Attendi 1 secondo prima di ripetere
}
```

Controllo di un LED:

Collegamento del LED:
Collega il catodo (negativo) del LED a un pin digitale di Arduino tramite una resistenza.
Collega l'anodo (positivo) del LED a 5V tramite una resistenza.
Esempio di Accensione e Spegnimento del LED:

```
int ledPin = 8;  // Pin digitale per controllare il LED

void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop() {
  digitalWrite(ledPin, HIGH);  // Accendi il LED
  delay(1000);  // Mantieni acceso il LED per 1 secondo
  digitalWrite(ledPin, LOW);  // Spegni il LED
  delay(1000);  // Attendi 1 secondo prima di ripetere
}
```

Esempio di uso Wi-Fi

Installare lib: WiFiNINA, DHT sensor library by Adafruit, PubSubClient by Nick

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <DHT.h>

const char* ssid = "crypto_lab"; // SSID WIFI
const char* password = "crypto_lab_2023"; // psw
const char* mqtt_server = "192.168.2.45"; // Indirizzo IP del server
MQTT
WiFiClient wifiClient;
PubSubClient client(wifiClient);

void setup() {
  Serial.begin(115200);
  Serial.begin(9600);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
}
```

```
void setup_wifi() {
  delay(10);
  Serial.println();
  Serial.print("Connessione a ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connesso");
  Serial.println("Indirizzo IP: ");
  Serial.println(WiFi.localIP());
}

void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
  delay(2000);
  float temperature = 50;
  float humidity = 50;
  if (isnan(temperature) || isnan(humidity)) {
    Serial.println("Errore nella lettura della temperatura/umidità!");
    return;
  }
  Serial.print("Temperatura: ");
  Serial.print(temperature);
  Serial.print(" °C, Umidità: ");
  Serial.print(humidity);
  Serial.println("%");

  String payload = String(temperature) + "," + String(humidity);
  char msg[100];
  payload.toCharArray(msg, 100);
  client.publish("sensor_data", msg);
  Serial.println("publish on sensor_data");
}

void reconnect() {
  while (!client.connected()) {
```

```
    Serial.print("Connessione al server MQTT...");
    if (client.connect("arduinoClient", "marco", "s3crEt")) {
      Serial.println("Connesso");
    } else {
      Serial.print("Fallito, rc=");
      Serial.print(client.state());
      Serial.println(" Riprovo fra 5 secondi");
      delay(5000);
    }
  }
}
```

`Installazioni necessarie per usare MQTT:`

Broker MQTT (RabbitMQ) RabbitMQ è un software open-source di message broker che originariamente implementava l'Advanced Message Queuing Protocol (AMQP) e da allora è stato esteso con un'architettura plug-in per supportare lo Streaming Text Oriented Messaging Protocol (STOMP), l'MQ Telemetry Transport (MQTT) e altri protocolli.

`How to install #RabbitMQ on windows 10 step by step`

Alcune caratteristiche chiave di RabbitMQ sono:

- **Consegna affidabile dei messaggi**: RabbitMQ assicura che i messaggi vengano consegnati ai destinatari previsti, anche in caso di guasti.
- **Scalabilità**: RabbitMQ può essere facilmente scalato verso l'alto o verso il basso per gestire le variazioni del volume dei messaggi.
- **Alta disponibilità**: RabbitMQ è altamente disponibile, il che significa che può sopportare guasti senza perdere i messaggi.
- **Flessibilità**: RabbitMQ supporta una varietà di protocolli di messaggistica e può essere utilizzato con un'ampia gamma di applicazioni.

Per lo scopo del progetto attuale sono necessari due plug-in:

- Management Plugin: Il plugin di gestione di RabbitMQ fornisce un'API basata su HTTP per la gestione e il monitoraggio di nodi e cluster RabbitMQ, insieme a un'interfaccia utente basata su browser e a uno strumento a riga di comando, rabbitmqadmin.
- Plugin MQTT: RabbitMQ supporta MQTT 3.1.1 tramite un plugin fornito nella distribuzione principale.

È possibile abilitare entrambi utilizzando il comando rabbitmq-plugins.
Utilizzando il cmd :

- C:\Program Files\RabbitMQ Server\rabbitmq_server-3.13.2\sbin>**rabbitmq-plugins enable rabbitmq_management**
- C:\Program Files\RabbitMQ Server\rabbitmq_server-3.13.2\sbin>**rabbitmq-plugins enable rabbitmq_mqtt**

Per avviare rabbitmq si può avviare sempre tramite cmd:

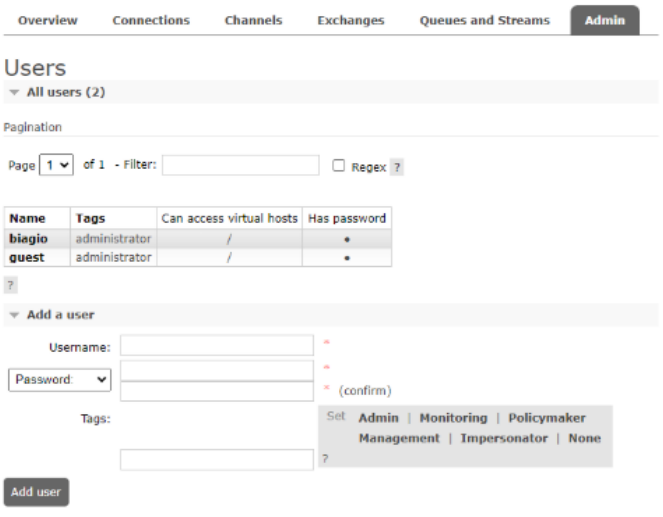
- C:\Program Files\RabbitMQ Server\rabbitmq_server-3.13.2\sbin>**rabbitmq-server.bat start**

L'interfaccia utente di gestione è accessibile tramite un browser Web all'indirizzo http://{node-hostname}:15672. L'interfaccia utente di gestione richiede l'autenticazione e l'autorizzazione, proprio come RabbitMQ la richiede ai client che si connettono. Oltre all'autenticazione, l'accesso all'interfaccia di gestione è controllato dai tag utente.
I tag sono gestiti tramite rabbitmqctl. Gli utenti appena creati non hanno alcun tag impostato per impostazione predefinita.

Dopo aver abilitato il plugin MQTT, anche il protocollo MQTT sarà disponibile per la comunicazione con il broker.
MQTT viene eseguito per impostazione predefinita sulla porta 1883; è possibile modificare la porta nel file di configurazione.
Le credenziali utilizzate per l'autenticazione sono diverse da quelle utilizzate per la gestione del pluigin. È possibile definire un nuovo utente utilizzando i seguenti comandi oppure crearlo tramite l'interfaccia utente.

```
# username and password are both "mqtt-test"
rabbitmqctl add_user mqtt-test mqtt-test
rabbitmqctl set_permissions -p / mqtt-test ".*" ".*" ".*"
rabbitmqctl set_user_tags mqtt-test management
```

È importante definire le autorizzazioni e i tag, per un buon livello di autorizzazione.
Nell'esempio, l'utente mqtt-test è in grado di leggere e scrivere su tutti gli argomenti ed è etichettato come utente di gestione.



Esempio recupero dati da un server MQTT:

composer.json:

```
{
    "require": {
        "bluerhinos/phpmqtt": "^1.0",
    }
}
```

index.php:

```
<?php
require __DIR__ . '/vendor/autoload.php'; // Includi il file
autoload.php di Composer

use Bluerhinos\phpMQTT;

$host = "localhost"; // Indirizzo IP del server MQTT
$port = 1883;
$username = "marco";
$password = "s3crEt";
$client_id = "altro";
$topic = "sensor_data";

$latest_temperature = "50";
$latest_humidity = "50";

function procmsg($topic, $msg) {
    global $latest_temperature, $latest_humidity;
    var_dump($msg);
    $data = explode(",", $msg);
    $latest_temperature = $data[0];
    $latest_humidity = $data[1];
}

$mqtt = new phpMQTT($host, $port, $client_id);
if(!$mqtt->connect(true, NULL, $username, $password)) {
    exit(1);
}

//$mqtt->subscribe($topic, 0, 'procmsg');
$topics[$topic] = array('qos' => 0, 'function' => 'procMsg');
$mqtt->subscribe($topics, 0);
```

```
$start_time = time();
while($mqtt->proc()) {
    if (time() - $start_time >= 2) {
        break;
    }
}

$mqtt->close();
?>

<!DOCTYPE html>
<html lang="en">
<head>
  <meta http-equiv="refresh" content="3">
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Arduino Sensor Data</title>
</head>
<body>
    <h1>Arduino Sensor Data</h1>
    <h2>Latest Data</h2>
    <p>Temperature: <?php echo $latest_temperature; ?> °C</p>
    <p>Humidity: <?php echo $latest_humidity; ?> %</p>
</body>
</html>
```

Descrizione:

Questo codice PHP si occupa di recuperare dati da un server MQTT (Message Queuing Telemetry Transport) e visualizzarli in una pagina web.
Spiegazione del codice:

1. Composer.json:
    - Questo file è utilizzato per gestire le dipendenze del progetto. Definisce le librerie PHP necessarie al progetto e le versioni compatibili.
    - `bluerhinos/phpmqtt` è una libreria PHP per la gestione delle connessioni MQTT.
2. Index.php:
    - Viene inclusa la libreria di Composer `autoload.php`, che si occupa di caricare automaticamente tutte le dipendenze definite nel file `composer.json`.
    - Vengono definiti le variabili per la connessione al server MQTT come l'indirizzo IP, la porta, il nome utente, la password e l'ID client.
    - Viene definito il topic MQTT da cui verranno ricevuti i dati dei sensori.
    - Viene definita una funzione `procmsg` che verrà chiamata quando un messaggio viene ricevuto dal topic MQTT. Questa funzione divide il messaggio in base al separatore "," e aggiorna le variabili `$latest_temperature` e `$latest_humidity` con i valori ricevuti.
    - Viene istanziato un oggetto `phpMQTT` che rappresenta la connessione al server MQTT. Se la connessione non riesce, il programma esce.
    - Viene sottoscritto il topic MQTT con la funzione `subscribe`.
    - Viene avviato un loop infinito che chiama `proc()` per gestire i messaggi MQTT ricevuti. Il loop si interrompe dopo 2 secondi dal suo avvio.
    - Infine, la connessione MQTT viene chiusa.
3. Pagina HTML:
    - Viene visualizzata una semplice pagina HTML che si aggiorna automaticamente ogni 3 secondi. (
    - Mostra i dati più recenti ricevuti dal server MQTT, ossia la temperatura e l'umidità.

Esempio salvataggio dati su DB:

```
<?php

// Parametri di connessione al database
$servername = "localhost";
$username = "root";
$password = "";
$database = "AmbientMonitoringDB";

// Connessione al database
$conn = new mysqli($servername, $username, $password);

// Controllo connessione
if ($conn->connect_error) {
    die("Connessione fallita: " . $conn->connect_error);
}

// Creazione del database se non esiste
$sql = "CREATE DATABASE IF NOT EXISTS $database";
if ($conn->query($sql) === TRUE) {
    echo "Database creato con successo o già esistente";
} else {
    echo "Errore nella creazione del database: " . $conn->error;
}

// Selezione del database
$conn->select_db($database);

// Creazione della tabella SensorData se non esiste
$sql = "CREATE TABLE IF NOT EXISTS SensorData (
    ID INT AUTO_INCREMENT PRIMARY KEY,
    Temperature FLOAT NOT NULL,
    Humidity FLOAT NOT NULL,
    Timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
)";
if ($conn->query($sql) === TRUE) {
    echo "Tabella SensorData creata con successo o già esistente";
} else {
    echo "Errore nella creazione della tabella SensorData: " .
$conn->error;
}

// Logica per salvare le informazioni dei sensori
```

```
function saveSensorData($temperature, $humidity) {
    global $conn;

    // Utilizzo di istruzioni preparate per evitare SQL injection
    $stmt = $conn->prepare("INSERT INTO SensorData (Temperature,
Humidity) VALUES (?, ?)");
    $stmt->bind_param("dd", $temperature, $humidity);

    if ($stmt->execute() === TRUE) {
        echo "Dati del sensore salvati con successo";
    } else {
        // Gestione degli errori
        echo "Errore nel salvataggio dei dati del sensore: " .
$conn->error;
    }

    $stmt->close();
}

// Esempio di utilizzo della funzione per salvare i dati del sensore
// saveSensorData(25.5, 60.0);

// Chiusura della connessione al database
$conn->close();

?>
```