PROGETTO PCTO

Descrizione

Lo scopo principale di questo progetto è quello di inviare dati relativi alla temperatura, umidità e batteria alla rete PAIOT di Lepida basata su LoRaWAN utilizzando la scheda Arduino MKR WAN 1310 e il sensore DHT22.

Procedimento

Step 1: Acquisizione dell'hardware

Arduino MKR WAN 1310

> L'arduino MKR WAN 1310 è una scheda compatta,avanzata e potente progettata per applicazioni LoRa ideale per progetti IoT che richiedono una comunicazione wireless a lunga portata a basso consumo energetico
>
> **APPLICAZIONI PRINCIPALI**
> - Monitoraggio ambientale
> - Tracciamento GPS
> - Agricoltura intelligente

> **CARATTERISTICHE PRINCIPALI**
> - Microcontrollore:SAMD21 Cortex-M0+ a 32 bit
> - Memoria: 256 KB di flash e 32 KB di ram
> - Modulo LoRa: Murata CMWX1ZZABZ,Chip SemtechSX1276
> - Interfacce: UART,SPI,I2C,8 ingressi analogici e 7 uscite digitali
> - Alimentazione: funzionamento tramite USB
> - Ottimizzato per dispositivi a basso consumo
> - Compatibile con il protocollo LoRaWAN, ideale per trasmissioni a lunga distanza

> **VANTAGGI**
> - Facile da programmare con l'IDE di arduino
> - Ideale per applicazione IoT
> - Consumo minore di energia
> - Compatibile con infrastrutture LoRaWAN

Sensore DHT22

> Il **DHT22** è un sensore digitale che misura temperatura e umidità con alta precisione, utilizzando un sensore capacitivo per l'umidità e un termistore per la temperatura.
>
> **ALIMENTAZIONE**
> 3,3 - 6 VCC
> **CONSUMO**
> In misura: 1,15 mA
> In standby: 50 µA

Step 2: Test dell'hardware

2.1: Comunicazione LoRa tra due dispositivi

Seguendo una guida nel sito di Arduino (https://docs.arduino.cc/tutorials/mkr-wan-1310/lora-send-and-receive/) abbiamo messo in comunicazione 2 dispositivi: uno che invia un messaggio con "Hello World!" e l'altro che lo riceve.
Con questo passaggio ci siamo assicurati che l'hardware per la comunicazione funziona correttamente.

2.2: Trovare i dati per la connessione alla rete di Lepida

Ci sono due modi principali per connettersi a qualsiasi rete LoRaWAN:

- OTAA (Over-The-Air Activation):
  - Il dispositivo invia una richiesta di join al server.
  - Utilizza l'**AppEUI**, il **DevEUI** e l'**AppKey** per autenticarsi.
  - Il server genera le chiavi di sessione **AppSKey** e **NwkSKey**.
- ABP (Activation By Personalization):
  - Le chiavi **AppSKey** e **NwkSKey** sono preconfigurate nel dispositivo.

Noi abbiamo deciso di utilizzare il primo metodo (OTAA), di conseguenza, come riportato precedentemente, abbiamo dovuto trovare/generare e utilizzare i seguenti valori:

- DevEUI (Device Extended Unique Identifier): È un identificativo che viene assegnato a ogni dispositivo durante la fabbricazione e non può essere modificato.
- AppKey: È una sequenza di 128 bit (32 caratteri esadecimali) che serve al dispositivo per generare un messaggio di join firmato da inviare al server. Il server successivamente verifica questo messaggio con l'AppKey dato in precedenza e se tutto va bene, vengono generate le chiavi di sessione.
- AppEUI: è un identificatore a 64 bit (8 byte) che rappresenta in modo univoco l'applicazione a cui il dispositivo appartiene.

Per trovare il **DevEUI**, abbiamo seguito una guida sul sito di Arduino (https://docs.arduino.cc/tutorials/mkr-wan-1310/the-things-network/) e abbiamo caricato uno sketch che mostra il DevEUI, una volta arrivati al quel punto, non abbiamo proseguito con la guida.
Per l'**AppKey**, visto che è una chiave segreta che deve essere salvata sia nel client che nel server, la abbiamo generata utilizzando il seguente sito: https://loratools.nl/#/keys.
Infine, per l'**AppEUI**, visto che Lepida non ci ha dato nessuna informazione a riguardo questo dato, ne abbiamo utilizzato uno generico: una sequenza di zeri lunga 8 bytes o 16 caratteri (**0000000000000000**).

2.3: Invio di dati di test

Una volta trovati tutti i dati necessari, ci siamo recati in un'area coperta dalla rete (la scuola) e abbiamo iniziato a inviare dei valori.
Abbiamo inizialmente provato a inviare dei semplici numeri, ma non venivano registrati bene visto che per ogni carattere compariva un "3" davanti.

28/03/2025 09:26:01	39
28/03/2025 08:59:08	3132
28/03/2025 07:20:39	35

Come si può vedere, abbiamo provato a inviare "5", "12", "9".
Abbiamo realizzato alla fine che era meglio inviare il tutto come una unica stringa in esadecimale, e infatti adesso i dati venivano registrati correttamente

03/04/2025 10:33:09	7012500a
03/04/2025 10:31:33	1013500a
03/04/2025 10:29:49	7012280a
03/04/2025 10:28:10	de12280a
03/04/2025 10:28:02	d412280a
03/04/2025 10:21:31	ca12320a
03/04/2025 10:21:05	7e13280a
03/04/2025 10:19:45	f2

Step 3: Collegamento del sensore & invio dei dati

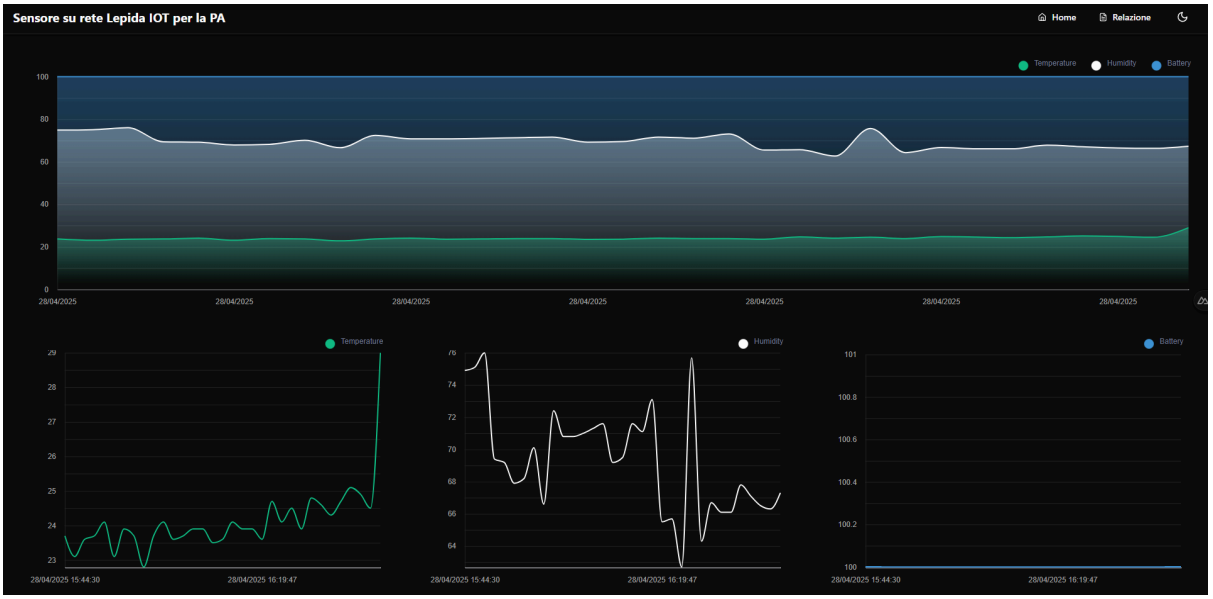
Una volta verificato che tutto fosse funzionante, abbiamo collegato il sensore e avviato l'invio dei dati.
Poiché gli output del sensore DHT22 sono numeri di tipo float (decimali) che occupano 4 byte, prima di inviarli a Lepida li moltiplichiamo per 100.
Questo passaggio consente di convertirli in numeri interi, mantenendo la precisione a due decimali, e riducendo la dimensione dei dati a soli 2 byte per valore.
In questo modo, siamo riusciti a trasmettere tutti e tre i dati (umidità, temperatura e batteria) all'interno di un payload di appena 6 byte.

Esempio: **4a1a6400540b**
Questo è un esempio di stringa inviata dal nostro Arduino, e tenendo in mente che abbiamo inviato 3 dati da 2 bytes ciascuno, nel formato **umidità** - **batteria** - **temperatura**, possiamo decodificarli nel seguente modo:

- Umidità: **4a1a** → **1a4a** (ruotare la stringa) → **6730** → **67,3**% (dividi per 100)
- Batteria: **6400** → **0064** (ruotare la stringa) → **100** → **100%**
- Temperatura: **540b** → **0b54** (ruotare la stringa) → **2900** -> **29,00°C** (dividi per 100)

Step 4: Interfaccia Web

Come ultima cosa, abbiamo creato una interfaccia web che utilizza le API di Lepida per ottenere i dati registrati e li mostra su dei grafici.
Il sito è accessibile all'indirizzo https://pcto-frontend.vercel.app.
Codice sorgente: https://github.com/le0o0oo/pcto-frontend



Il sito è stato fatto con il framework Nuxt.

I dati inviati dalla scheda vengono reperiti nel backend del sito con la seguente funzione:

```
async function fetchData() {
  consola.info("Fetching data from Lepida API...");
  const res = await $fetch(
    `https://api.retepaiot.it/sensordata/${config.app_eui}`,
    {
      method: "POST",
      body: {
        auth_key: config.lepida_api_key,
        from: config.start_from,
      },
    }
  );

  cache = res;
  consola.success("Data fetched & cached");
}
```

Vengono poi decodificati in questo modo:

```ts
// decoder.ts
function rotateString(str: string, n: number): string {
  n = n % str.length;
  return str.slice(n) + str.slice(0, n);
}

export default (
  input: string
): { temp: number; battery: number; hum: number } => {
  let temp, hum, battery;

  let humHex = input.slice(0, 4);
  let batteryHex = input.slice(4, 8);
  let tempHex = input.slice(8, 12);

  humHex = rotateString(humHex, 2);
  tempHex = rotateString(tempHex, 2);
  batteryHex = rotateString(batteryHex, 2);

  temp = parseInt(tempHex, 16) / 100;
  hum = parseInt(humHex, 16) / 100;
  battery = parseInt(batteryHex, 16);

  return { temp, battery, hum };
};
```