Lab 2

Communication Internet et Bluetooth avec la carte Arduino Nano

2.1.Serveur WEB avec la lecture du capteur type DHT (11/22)

Dans notre premier exemple nous déployons un simple serveur WEB connecté sur le réseau IP par le biais d'une interface Ethernet. Pendant une connexion avec le serveur demandée par un client connecté sur le même réseau le serveur active le capteur type DHT11 pour lire et ensuite envoyer les données au client.
Les données sont envoyées dans une page WEB (HTML).

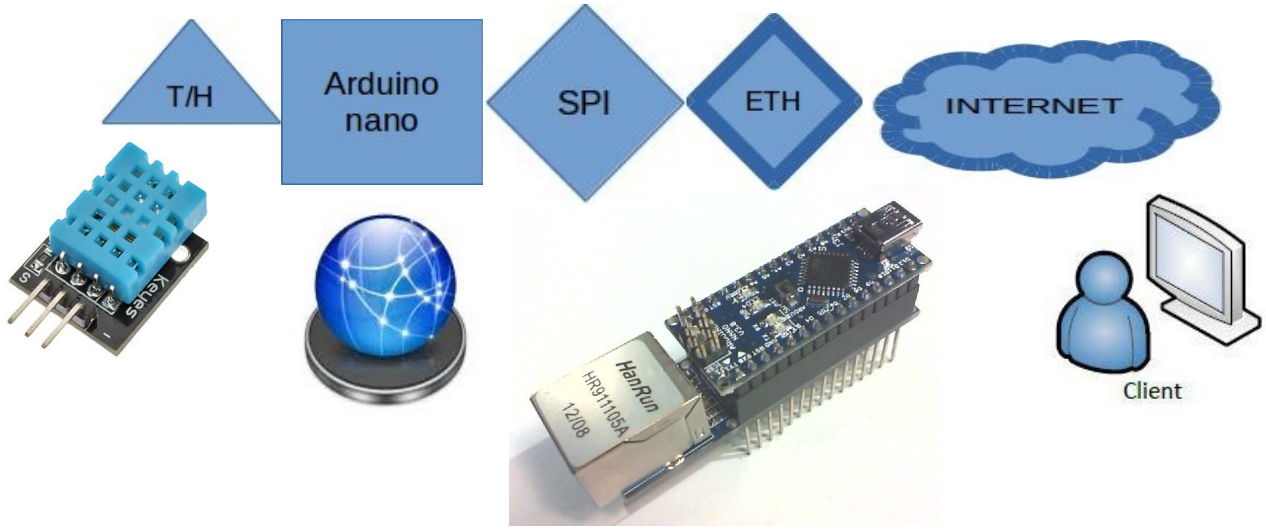

Figure 2.1 Architecture symbolique avec capteur T/H et Arduino Nano avec un shield Ethernet.

```
#include <SPI.h>
#include <UIPEthernet.h> // Used for Ethernet
#include "DHT.h"
#define DHTPIN 5     // what digital pin we're connected to
#define DHTTYPE DHT11   // DHT 11
// **** ETHERNET SETTING ****
byte mac[] = { 0x54, 0x34, 0x41, 0x30, 0x30, 0x31 };
IPAddress ip(192, 168, 1, 179);
EthernetServer server(80);
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
  // start the Ethernet connection and the server:
  Ethernet.begin(mac, ip);
  server.begin();
  Serial.print("IP Address: ");
  Serial.println(Ethernet.localIP());
  dht.begin();
}
```

```
void loop() {
  // listen for incoming clients
  EthernetClient client = server.available();
  if (client)
  {
    Serial.println("-> New Connection");
    // an http request ends with a blank line
    boolean currentLineIsBlank = true;
    while (client.connected())
    {
      if (client.available())
      {
        char c = client.read();
        // if you've gotten to the end of the line (received a newline
        // character) and the line is blank, the http request has ended,
        // so you can send a reply
        if (c == '\n' && currentLineIsBlank)
        {
          char page[256];
          float h = dht.readHumidity();
          float t = dht.readTemperature();
          sprintf(page,"..>",..);
          client.println(page);
          break;
        }
        if (c == '\n') {
          // you're starting a new line
          currentLineIsBlank = true;
        }
        else if (c != '\r')
        {
          // you've gotten a character on the current line
          currentLineIsBlank = false;
        }
      }
    }
    // give the web browser time to receive the data
    delay(10);
    client.stop();      // close the connection
    Serial.println("   Disconnected\n");
  }
}
```

2.1.1 Le bus et le protocole SPI

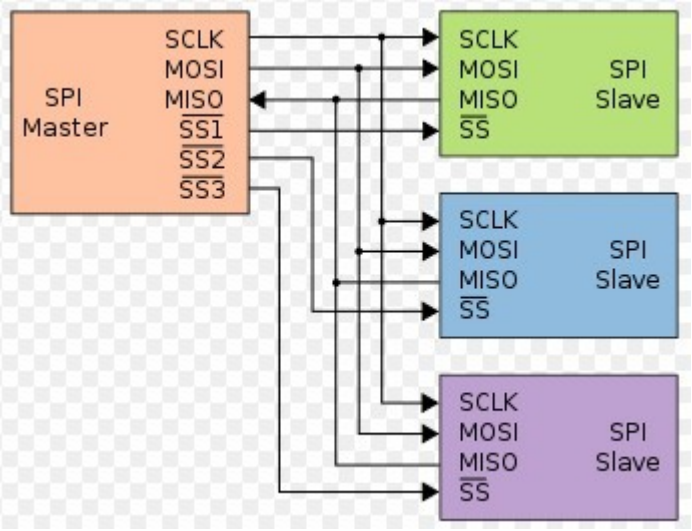

Figure 2.2 Liaison SPI avec un maître et trois esclaves

Le bus SPI utilise **quatre signaux logiques** :

- **SCLK** Serial Clock, Horloge (généré par le maître)
- **MOSI** Master Output, Slave Input (généré par le maître)
- **MISO** Master Input, Slave Output (généré par l'esclave)
- **SS** Slave Select, Actif à l'état bas (généré par le maître)

Une transmission SPI typique est une communication simultanée entre un maître et un esclave :

- Le maître génère l'horloge et sélectionne l'esclave avec qui il veut communiquer par via le signal SS
- L'esclave répond aux requêtes du maître

À chaque coup d'horloge le maître et l'esclave s'échangent un bit. Après huit coups d'horloges le maître a transmis un octet à l'esclave et vice versa. La vitesse de l'horloge est réglée selon des caractéristiques propres aux périphériques.

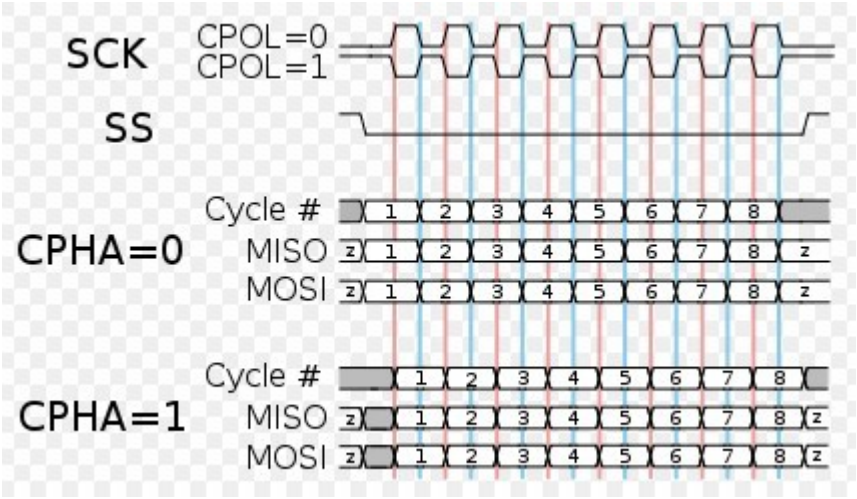

Figure 2.3 Protocole SPI entre un maître et un esclave

Avantages

- Communication Full duplex
- Débit plus important qu'un bus I^2C
- Flexibilité du nombre de bits à transmettre ainsi que du protocole en lui-même
- Simplicité de l'interface matérielle

    - Aucun arbitre nécessaire car **aucune collision possible**
    - Les esclaves utilisent l'**horloge du maître** et n'ont donc pas besoin d'oscillateur propre
    - Pas de phy nécessaire
- Partage d'un bus commun pour l'horloge, MISO et MOSI entre les périphériques Inconvénients

Inconvénients

- Monopolise plus de broches d'un boîtier que l'I^2C ou une UART qui en utilisent seulement deux.
- Aucun adressage possible, il **faut une ligne de sélection par esclave** en mode non chaîné.Communication Internet et Bluetooth avec la carte Arduino Nano
- Le protocole n'a pas d'acquittement. Le maître peut parler dans le vide sans le savoir.
- La plupart des implémentations ne tolèrent la présence que d'un seul maître SPI sur le bus.

Exercice :

Préparer la page WEB (une ligne) avec les arguments a envoyer vers le client.

2.2 Clients WEB de ThingSpeak

2.2.1 Serveur ThingSpeak

ThingSpeak est une application open source d' »Internet of Things » (IoT) et une API pour stocker et récupérer des données provenant de l'utilisation du protocole HTTP sur Internet ou via un réseau local. ThunderSpeak a été initialement lancé par ioBridge en 2010 comme un service à l'appui des applications IoT. ThingSpeak permet la création des applications de journalisation des données provenant de capteurs, des applications de suivi de la localisation .

ThingSpeak.com (Matlab) a intégré le support du logiciel informatique numérique de MathWorks. Il permettre aux utilisateurs de ThingSpeak.com d'analyser et de visualiser les données téléchargées à l'aide de Matlab sans nécessiter l'achat d'une licence Matlab de Mathworks.
Le service de base est gratuit limitant la fréquence d'accès à la base à un accès toutes les 15 secondes. Les services offrant les fréquences d'accès plus élevées sont payants.

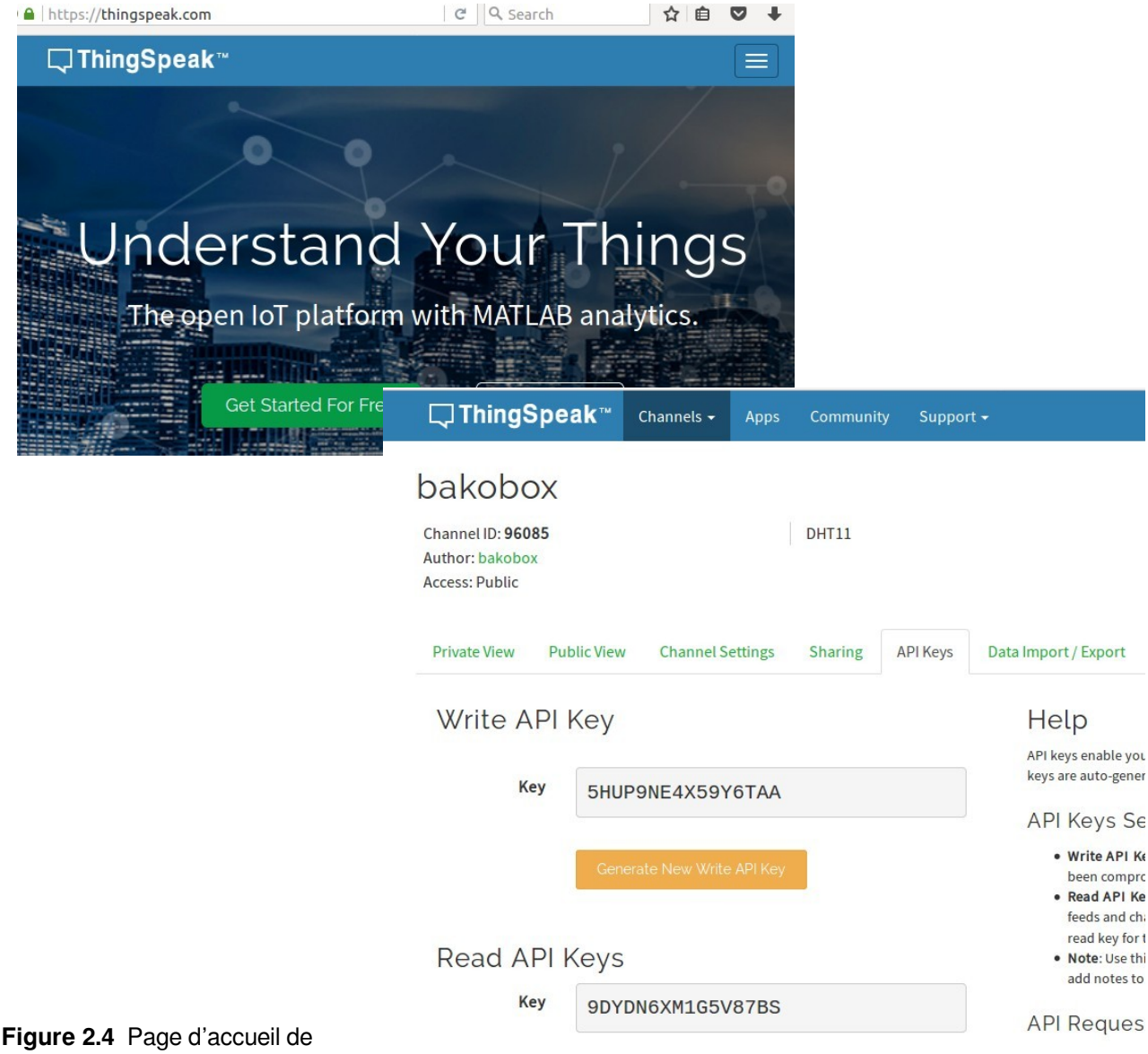

Figure 2.4 Page d'accueil de **ThingSpeak.com** et l'exemple d'un ***channel*** avec son identificateur ID et les clés d'accès en écriture et en lecture

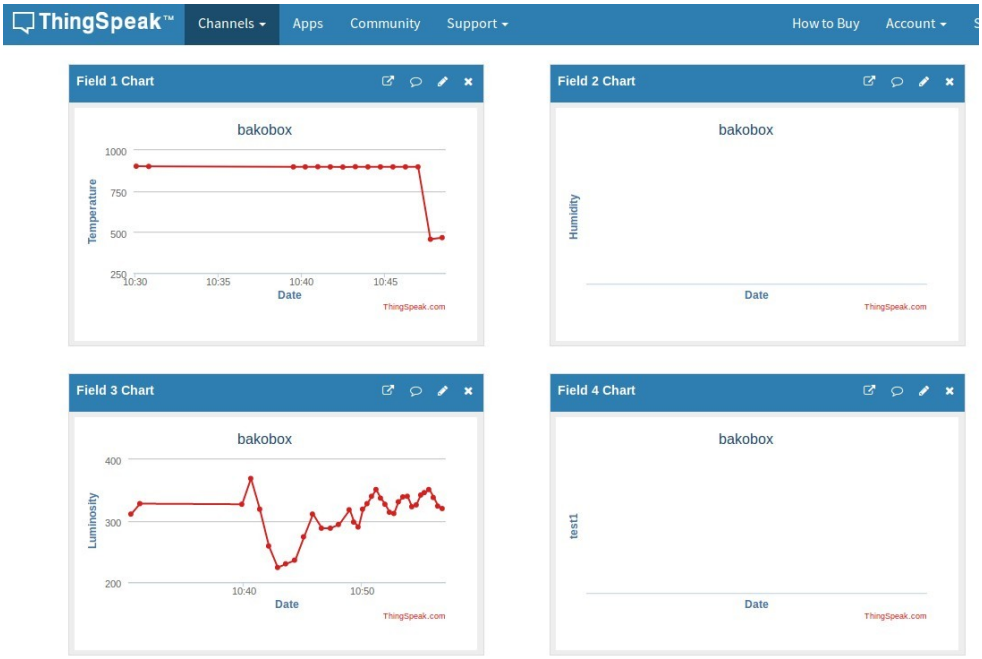

Figure 2.5 Visualisation des chronogrammes du ***channel*** bakobox (ID:96085) avec un maximum de 8 ***fields***

L'envoi des données sur le serveur **ThingSpeak** consiste à préparer une requête HTML de type **`GET`** ou **`POST`** , elle implique l'utilisation de la clé d'écriture:
Exemple de **`GET`** :

```
GET https://api.thingspeak.com/update?api_key=5HUP9NE4X59Y6TAA&field1=0
```

La réception des données nécessite la connaissance du numéro du ***channel*** (ID) et le numéro du ***field***:

```
GET https://api.thingspeak.com/channels/96085/fields/1.json?results=2
```

Attention : les résultat est récupéré en format **`JSON`** (deux enregistrements)

2.2.1 Client WEB de ThingSpeak.com avec capteur DHT (version proxy)

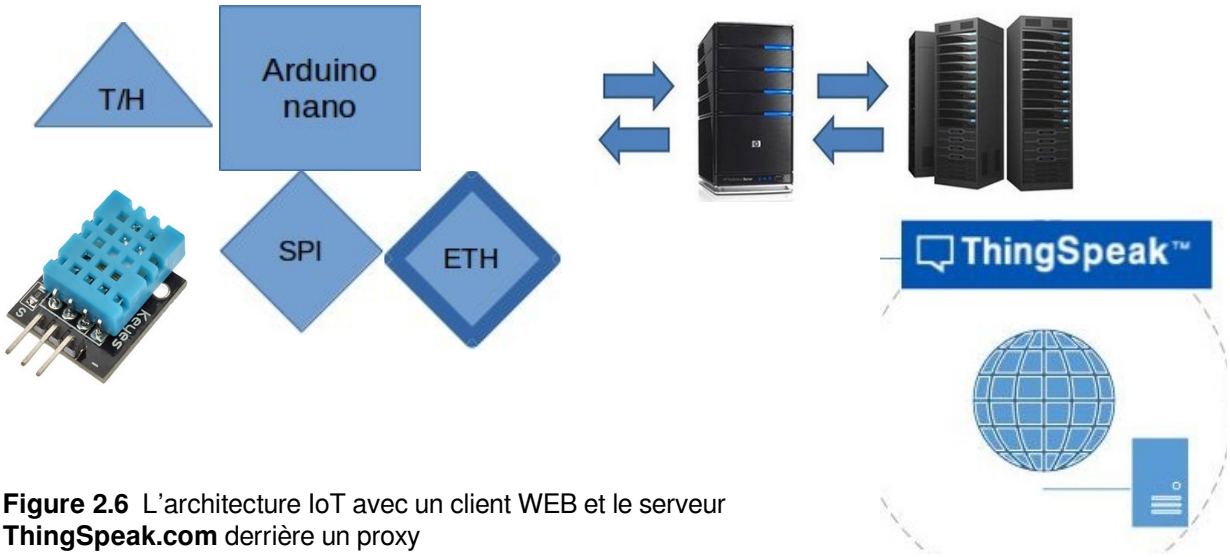

Figure 2.6 L'architecture IoT avec un client WEB et le serveur **ThingSpeak.com** derrière un proxy

Le code Arduino pour la connexion d'une carte Nano avec extension Ethernet et un capteur DHT11/22 et l'envoi des données vers **`ThingSpeak.com`** avec la prise un compte du serveur proxy de l'école (réseau étudiants) **`(193.52.104.30:3128).`**

La requête HTML est envoyée en format POST ; les données sont envoyées après la création de l'entête comme le contenu de la page.

```
#include <SPI.h>
#include <UIPEthernet.h>
#include "DHT.h"
#define DHTPIN 5     // what digital pin we're connected to
#define DHTTYPE DHT11   // DHT 11
byte mac[] = { 0xDE, 0xFA, 0x12, 0xAA, 0xFC, 0xED };  // to modify
byte ip[] ={172,19,65,60};
byte dnsa[] = {172,19,0,4};
byte gateway[]= {172,19,64,3};
byte subnet[] = {255,255,248,0};
byte proxy[] = {193,52,104,30};
EthernetClient client;
char *key="P18HVHSXTYAARD83";
int d1=0,d2=0,f1=0,f2=0;
char PostData[100]; //= "api_key=77QWEFDSMCU3TMGU&field4=42.0&field5=82.97";

DHT dht(DHTPIN, DHTTYPE);

int TSupdate(char *data)
{
  String str(data);
  Serial.println("connecting.");
  if (client.connect("193.52.104.30",3128)) { // proxy
  client.println("POST http://184.106.153.149/update HTTP/1.1");
  client.println("Host: 172.19.65.60");
  client.println("User-Agent: Arduino/1.0");
  client.println("Connection: close");
  client.print("Content-Length: ");
  client.println(str.length());
  client.println();
  client.println(str);
  } else {
    Serial.println("connection failed");
  }
  delay(2000);
  Serial.println("disconnecting.");
  client.stop();
}

void dhtread()
{
          float h = dht.readHumidity();
          delay(200);
          float t = dht.readTemperature();
          delay(200);
          d1=(int)t;f1=0;d2=(int)h;f2=0;
}

void setup() {
  Serial.begin(9600);
  Ethernet.begin(mac,ip,dnsa,gateway,subnet); // use with proxy
  //Ethernet.begin(mac,ip); // use with proxy
  delay(1000);Serial.print("IP = ");
  Serial.println(Ethernet.localIP());
  Serial.println("first update");
  dht.begin();
  dhtread();
```

```
  sprintf(PostData,"api_key=%s&field1=%02d.%02d&field2=%02d.
%02d",key,d1,f1,d2,f2);
  TSupdate(PostData);
}

void loop() {
  if (client.available()) {
    char c = client.read();
    Serial.print(c);}
    delay(60000);
    Serial.println("next update");
    dhtread();
    sprintf(PostData,"api_key=%s&field1=%02d.%02d&field2=%02d.
%02d",key,d1,f1,d2,f2);
    TSupdate(PostData);
  }
```

2.2.1 Client WEB de ThingSpeak avec capteur analogique (version sans proxy)

Dans l'exemple précédent nous avons préparé notre page WEB « manuellement » ce qui nous donne un code très compact. Pour faciliter l'utilisation des fonctions de communication avec le serveur ThingSpeak nous pouvons exploiter une bibliothèque spécifique **`ThingSpeak.h`**. Le code de cette bibliothèque sera ajouté dans notre programme. Dans le cas d'une carte Arduino Nano il nous reste peu de mémoire pour les bibliothèques des **capteurs numériques**.
Pour tester les fonctions de la bibliothèque ThingSpeak.h nous utiliserons donc un **capteur analogique** de la luminosité. D'autre part l'utilisation de cette bibliothèque ne permet pas de communiquer avec le serveur ThingSpeak via un serveur proxy.
La solution est de communiquer avec un serveur ThingSpeak installé sur le réseau local.

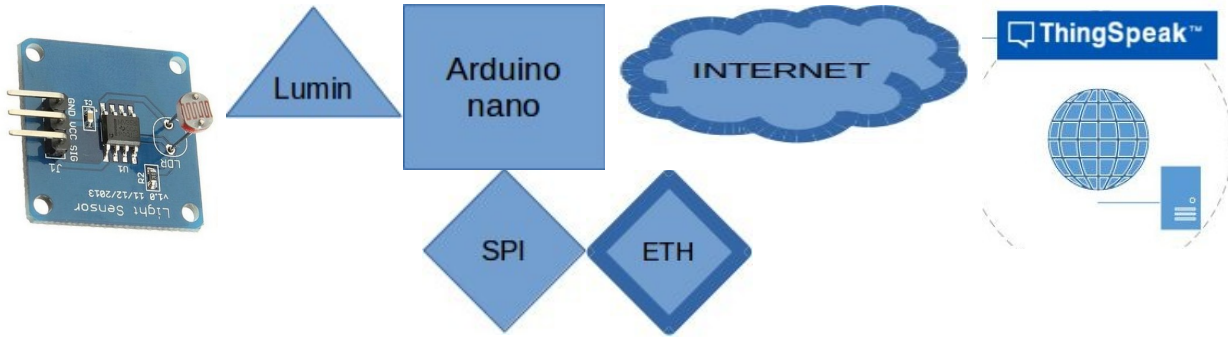

Figure 2.7 L'architecture IoT avec un client WEB (bibliothèque ThingSpeak.h) et le serveur **ThingSpeak** sur le réseau local

```
#include <SPI.h>
#include <UIPEthernet.h>
#include "ThingSpeak.h"
byte mac[] = { 0xDE, 0xFA, 0x12, 0xAA, 0xFC, 0xED };  // to modify
byte ip[] ={192,168,1,80};  // Ethernet connexion
EthernetClient client;
const char *apiKey = "5HUP9NE4X59Y6TAA";
unsigned long chNum = 96085;
int photocellPin = 0;     // the cell and 10K pulldown are connected to A0
int photocellReading;     // the analog reading from the sensor divider

void setup() {
  Serial.begin(9600);
  Ethernet.begin(mac,ip); // no proxy
  delay(1000);
```

```
  Serial.println(Ethernet.localIP());
  ThingSpeak.begin(client);
  delay(300);
}
void loop() {
    photocellReading = analogRead(photocellPin);
    ThingSpeak.writeField(chNum,3,photocellReading,apiKey);
    delay(20000);
  }
```

2.3 Carte Arduino Nano avec un modem Bluetooth (HC-05 ou HC-06)

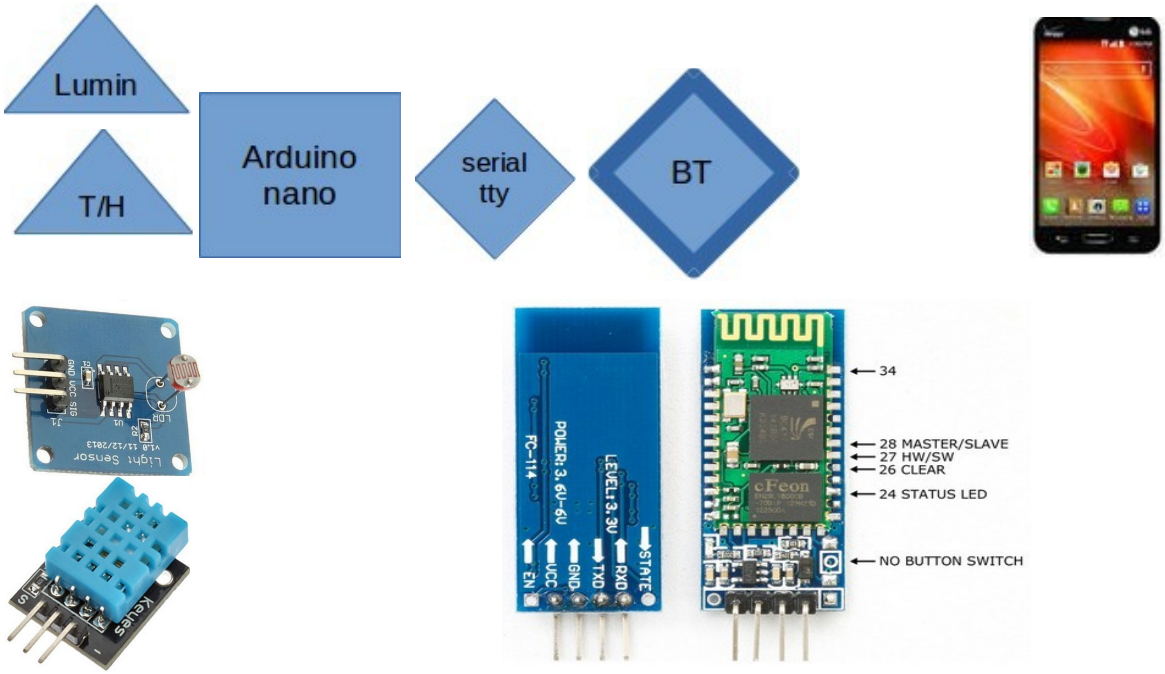

Figure 2.8 L'architecture IoT avec un serveur Bluetooth sur la carte Arduino Nano et un client **Bluetooth terminal** sur Android

L'utilisation de cet exemple nécessite téléchargement d'une applicationAndroid sur votre smartphone (Bluetooth terminal) puis l'activation du Bluetooth et ***pairing*** avec HC-06 (HC-05).
Le code Arduino exploite la biblithèque SoftwareSerial.h qui permet de mettre en ouvre le protocole UART sur les broches sélectionnées dans **`SoftwareSerial BTserial(2, 3);`** . La sortie **TX** de HC-06 va vers la broche **D2 (Rx)** de la carte, et l'entrée **RX** de HC-06 va vers la broche **D3(Tx)** de la carte. La ligne VCC est connecté sur 5V.

2.3.1 Le code Arduino de l'architecture 2.3

```
#include <SoftwareSerial.h>
#include "DHT.h"
#define DHTPIN 5     // what digital pin we're connected to ?
#define DHTTYPE DHT11   // DHT 11
SoftwareSerial BTserial(2, 3); // RX | TX
DHT dht(DHTPIN, DHTTYPE);
String message; //string that stores the incoming message
char temp[16];
float t,h;

int  readth()
 {
 delay(500);
 h = dht.readHumidity(); // a completer
 t = dht.readTemperature();
if (isnan(h) || isnan(t)) {
    Serial.println("Failed to read from DHT sensor!"); return 1;
  }
  return 0;
 }
```

```
void setup()
{
  Serial.begin(9600); //set baud rate
  BTserial.begin(9600);
  dht.begin();
  }
void loop()
{
  while(BTserial.available())
    message+=char(BTserial.read()); //store string from serial command
  if(!BTserial.available())
  {
    if(message!="")
      Serial.println(message); //show the data
      readth();
      switch(message[0])
        {
          case 't': sprintf(temp," T:%d°C",(int)t);
                    Serial.println(temp);
                    BTserial.print(temp); break;
          case 'h': sprintf(temp," Hrel:%d%",(int)h);
                    Serial.println(temp);
                    BTserial.print(temp); break;
// a completer
          default: sprintf(temp," t,h,l only");
                    Serial.println(temp);
                    BTserial.print(temp);
        }
      message=""; //clear the data
    }
  }
  delay(500); //delay
}
```

2.4 Exercice :

Ajouter capteur de la luminosité dans l'architecture 2.3, compléter le code et tester l'ensemble sur un smartphone.