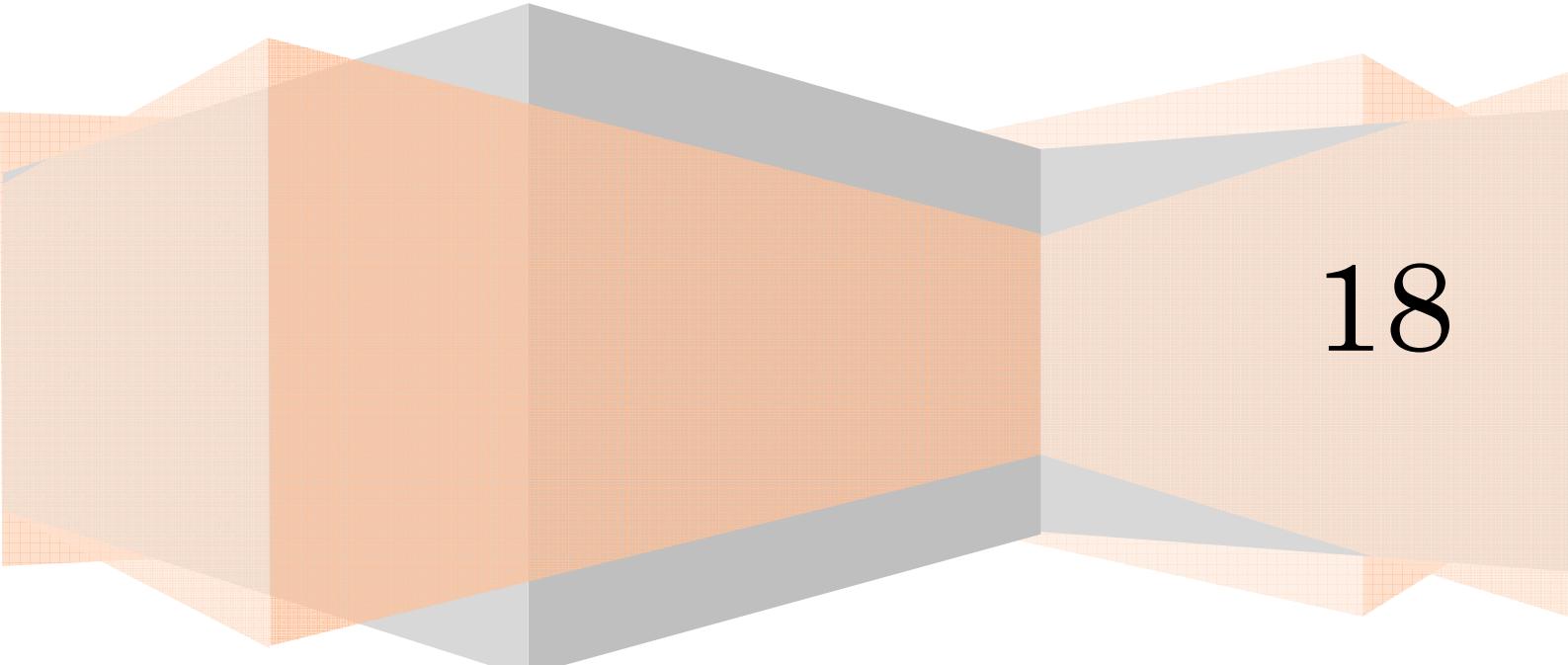


Lycée Raymond Queneau

Projet final BTS SN

La maison du futur - Domotique

Chauveau Aurélien

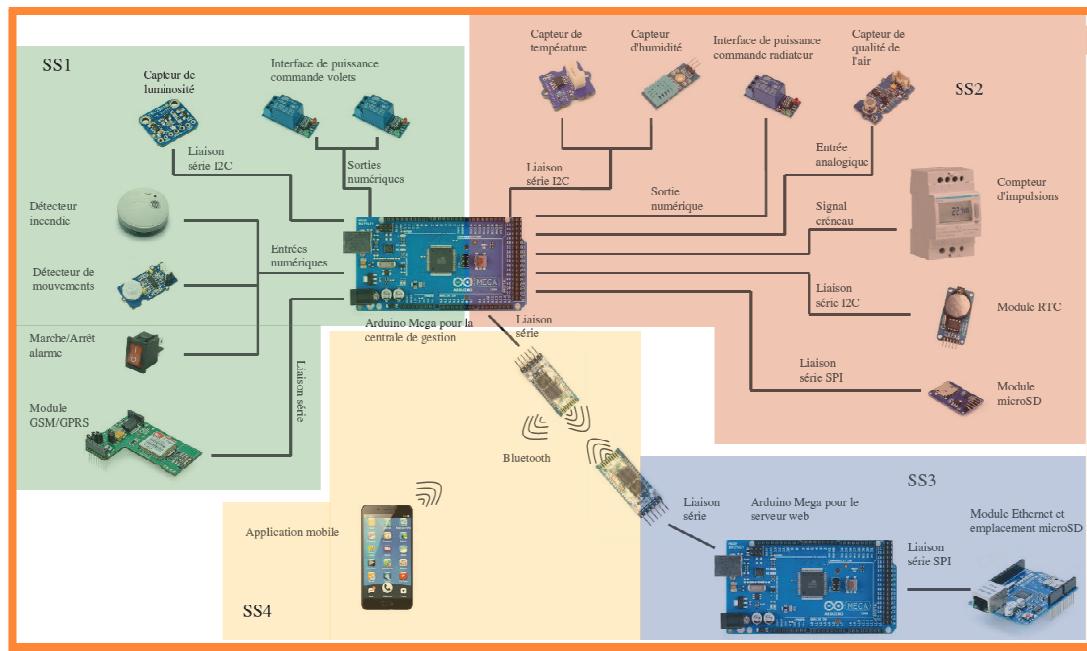


18

SOMMAIRE

1 - Présentation.....	p3
2 - Les tests unitaires.....	p5
Le thermomètre.....	p5
L'hygromètre.....	p6
Capteur de qualité d'air.....	p8
Carte SD.....	p10
Radiateur.....	p12
Module RTC.....	p14
Compteur Electrique.....	p15
3 - Regroupement des composants.....	p17
Le compteur électrique, la carte SD et le module RTC.....	p17
Le thermomètre,l'hygromètre, capteur de qualité d'air et le radiateur.....	p20
Regroupement final des composants.....	p21
4 - Regroupement des différentes parties.....	p27
La bibliothèque capteur.h.....	p29
La bibliothèque gestionMaison.h.....	p30
Fonction d'envoi des trames.....	p31

1 - PRÉSENTATION



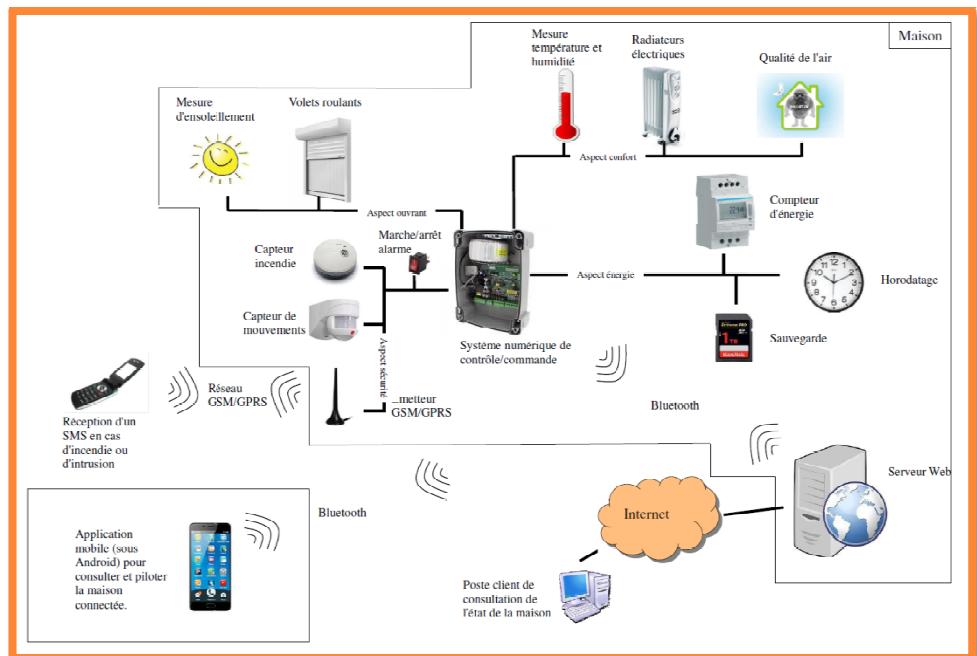
Le projet domotique (la maison du futur), consiste à automatiser une maison. Pour réaliser cela, nous sommes 4 étudiants, répartis sur différents aspects du projet.

L'étudiant 1 se chargera du contrôle des volets, de la lumière et de la partie sécurité qui comprend un capteur de mouvement et de fumée.

L'étudiant 3 sera chargé de créer un serveur web qui affichera les valeurs envoyées par les étudiants 1 et 2.

L'étudiant 4 devra créer une application mobile qui permettra de contrôler les actionneurs des étudiants 1 et 2 et d'afficher les valeurs envoyées par les étudiants 1 et 2.

Les étudiants 3 et 4 auront une partie commune sur la communication Bluetooth qui servira à la communication entre la centrale de gestion (carte Arduino), l'application et le serveur web.

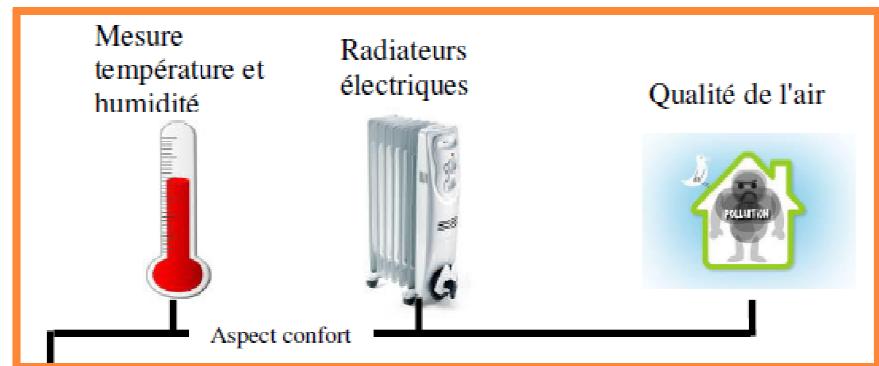




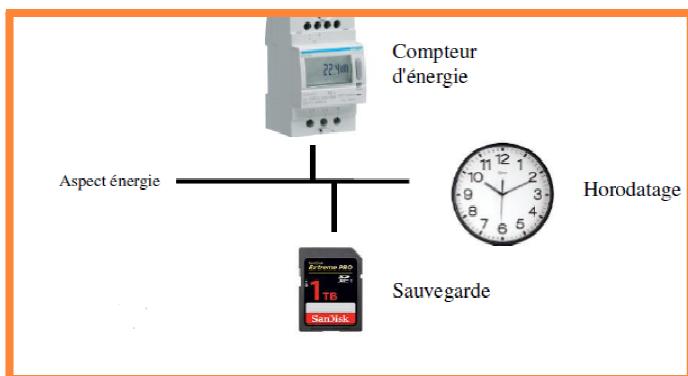
En tant qu'**étudiant 2**, je suis chargé de l'aspect confort et consommation.

La partie confort du projet consiste à récupérer plusieurs valeurs, utiles au propriétaire, telles que : la température, le taux d'humidité ou encore la qualité de l'air. Nous aurons pour cela différents capteurs à choisir et à mettre en marche afin de réaliser cette collecte des valeurs.

Nous devrons également piloter un relais via une carte Arduino. Ce dernier nous permettra par la suite de contrôler et d'automatiser un radiateur. Pour automatiser le radiateur nous devrons récupérer une valeur (la température), saisie par l'utilisateur, puis nous devrons égaliser la température de la maison à la valeur récupérée. Une plage d'hystérésis est donc nécessaire, elle sera de 1°C.



Chaque valeur récupérée devra être envoyée à intervalle régulier au serveur web et à l'application mobile. La température et l'humidité devront être envoyées toutes les 5 secondes et la qualité de l'air toutes les 10 secondes.



L'aspect consommation d'énergie est aussi à ma charge. Pour se faire nous avons un compteur d'énergie, une horloge et une carte SD. Tout cela devra nous permettre d'horodater¹ et de sauvegarder sur une carte SD la consommation en watt par heure.

Une partie commune entre l'étudiant 1 et 2 nous charge de regrouper nos codes dans une seule carte Arduino et de créer un objet pour stocker le maximum de valeur utile au bon fonctionnement du projet. Le code final devra être lisible, intuitif et donc surtout compréhensible.

Nous devrons aussi réunir tous les étudiants du groupe afin de nous mettre d'accord sur le protocole de la trame de données à envoyer au serveur web et à l'application.

¹**Horodater** :Ajouter de manière automatique la date et l'heure à un document.

2 - LES TESTS UNITAIRES

LE THERMOMÈTRE :

Le capteur de pression BMP180 étant équipé d'un capteur de température, nous l'avons utilisé pour notre projet. J'ai par la suite récupéré une bibliothèque sur GITHUB² pour commander le capteur. J'ai, ensuite, relié le BMP180 à une carte Arduino, créé un code simple et inclus la bibliothèque à mon code pour réaliser des tests. La communication entre la carte Arduino et notre capteur se fait par I2C.

Code de test unitaire	Valeur obtenue	Montage
<pre>thermometre § #include <SFE_BMP180.h> #include <Wire.h> #include "SFE_BMP180.h" SFE_BMP180 pressure; void setup() { Serial.begin(9600); //lancement du programme d'initialisation if (pressure.begin()) Serial.println("initialisation BMP180 reussi"); else { Serial.println("initialisation BMP180 echouer\n\n"); while(1); } } void loop() { double T; //lecture des données de température pressure.startTemperature(); delay(5); //récupération pressure.getTemperature(T); //affichage Serial.print("temperature: "); Serial.print(T,2); Serial.println(" deg C"); delay(3000); }</pre>	<pre>temperature: 21.07 deg C temperature: 21.07 deg C temperature: 21.06 deg C temperature: 21.05 deg C temperature: 21.05 deg C temperature: 21.06 deg C temperature: 21.05 deg C temperature: 21.04 deg C temperature: 21.04 deg C temperature: 21.05 deg C temperature: 21.05 deg C temperature: 21.07 deg C temperature: 21.05 deg C temperature: 21.05 deg C temperature: 21.04 deg C</pre>	

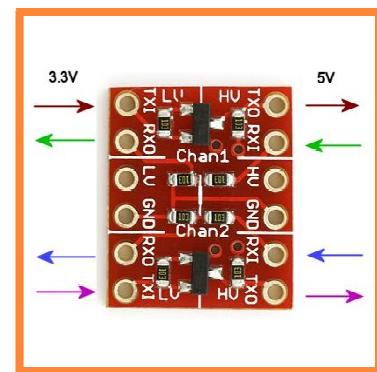
²GITHUB : Service web d'hébergement et de gestion de développement de logiciels.

L'HYGROMÈTRE :

Un hygromètre est un capteur qui sert à mesurer le taux d'humidité dans l'air. Pour se faire nous allons utiliser le capteur HIH6130 qui sert aussi de thermomètre et une bibliothèque trouvée sur GITHUB pour gagner du temps. Le HIH6130 mesure le taux d'humidité à plus ou moins 5% entre 10% et 90% ainsi que la température entre 5°C et 50°C. Nous allons aussi avoir besoin d'un convertisseur de tension, qui nous servira à ne pas cramer notre capteur d'humidité/température avec le bus I2C qui fonctionne en 5V alors que notre HIH6130 reçoit du 3,3V.

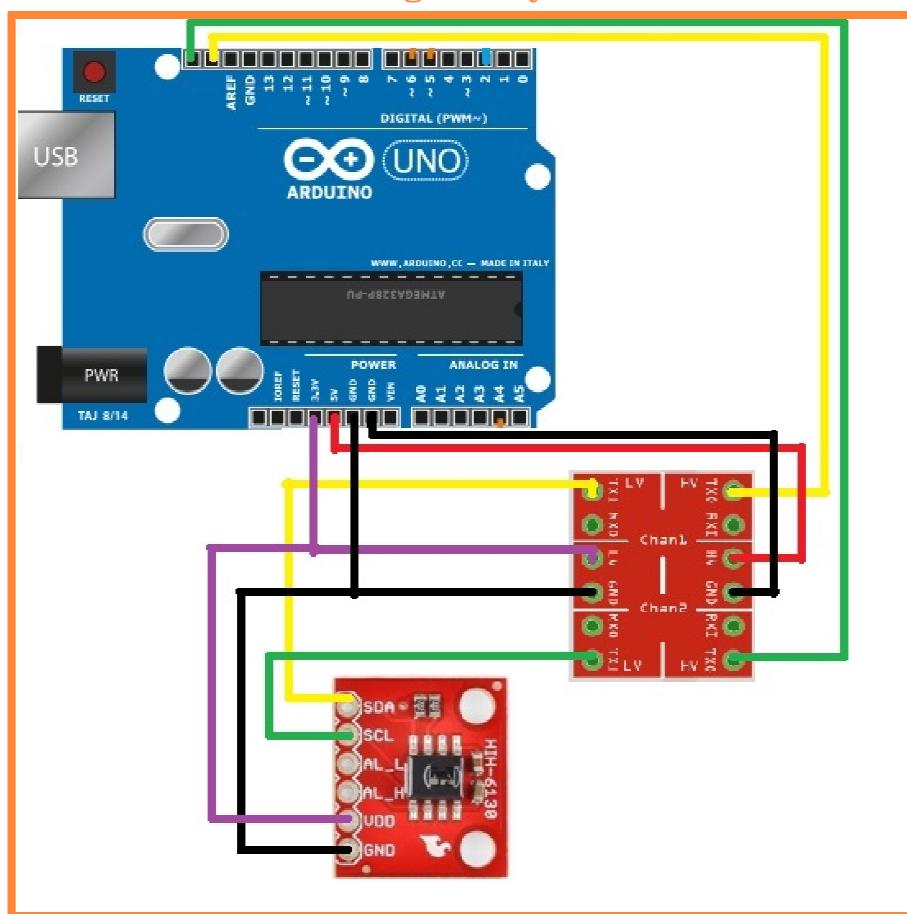


HIH6130



Convertisseur

Montage du système



Le convertisseur nous permet de passer de 5V à 3,3V pour passer de la carte Arduino au capteur et vice-versa.

Le code de test unitaire m'a permis de vérifier les valeurs de température du capteur, qui sont correctes. Nous allons donc plutôt utiliser le HIH6130 pour pouvoir à la fois récupérer la température et le taux d'humidité, au lieu d'avoir deux capteurs distincts.

Code de test unitaire

HIH6130tester

```
#include <Wire.h>
#include <HIH6130.h>

// L'adresse du capteur est 0x27
HIH6130 rht(0x27);

void setup(){
    Serial.begin(9600);
    //on lance le programme RHT
    //Read Humidity Temperature"
    rht.begin();
}

void loop(){
    // lecture des information
    //qui seront mise dans l'objet rht
    rht.readRHT();

    //Affichage des informations
    //hummidité et temperature
    //de l'objet rht
    Serial.print(rht.humidity);
    Serial.println ("%");
    Serial.print(rht.temperature);
    Serial.println (" Celsius");

    delay(2000);
}
```

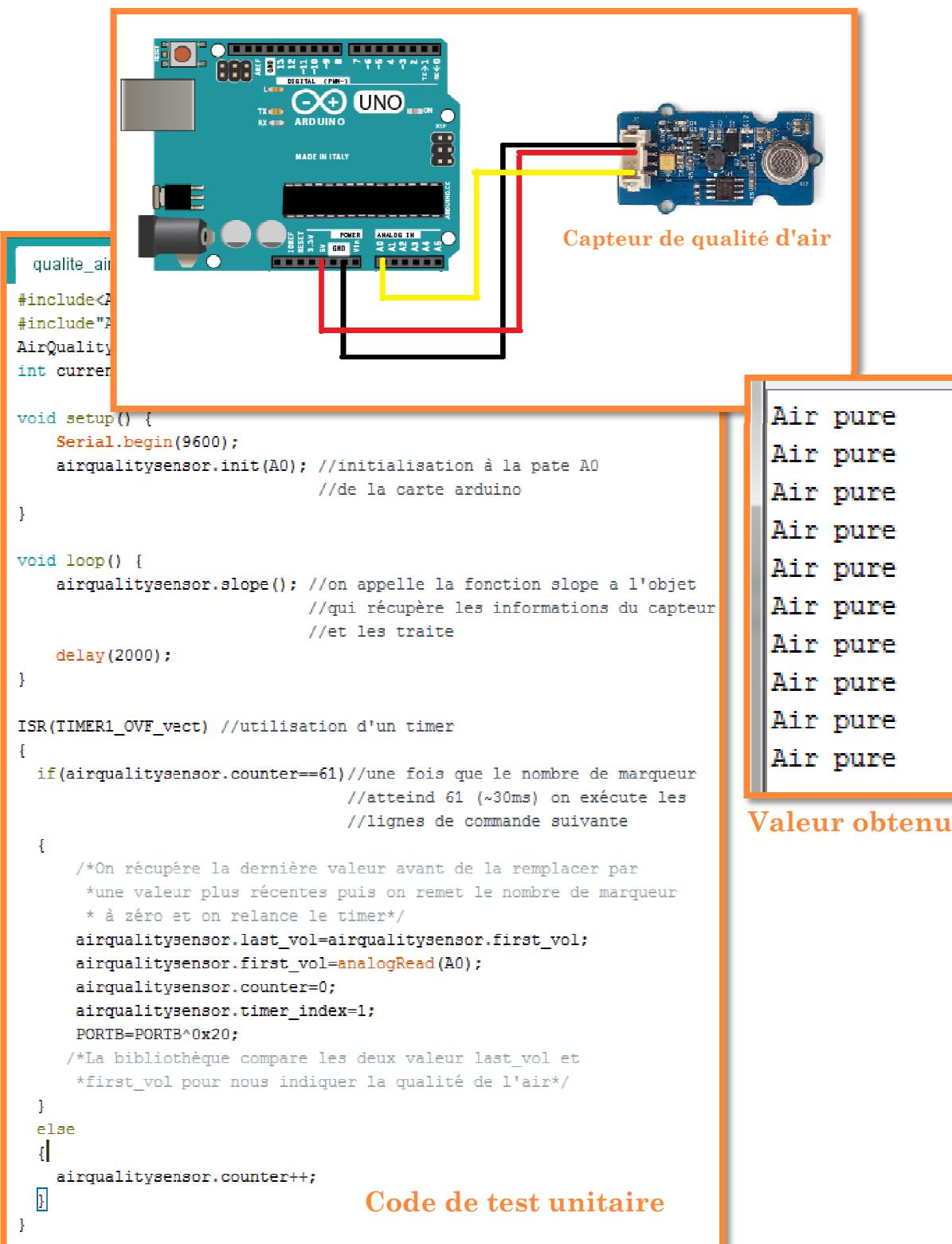
Valeur obtenue

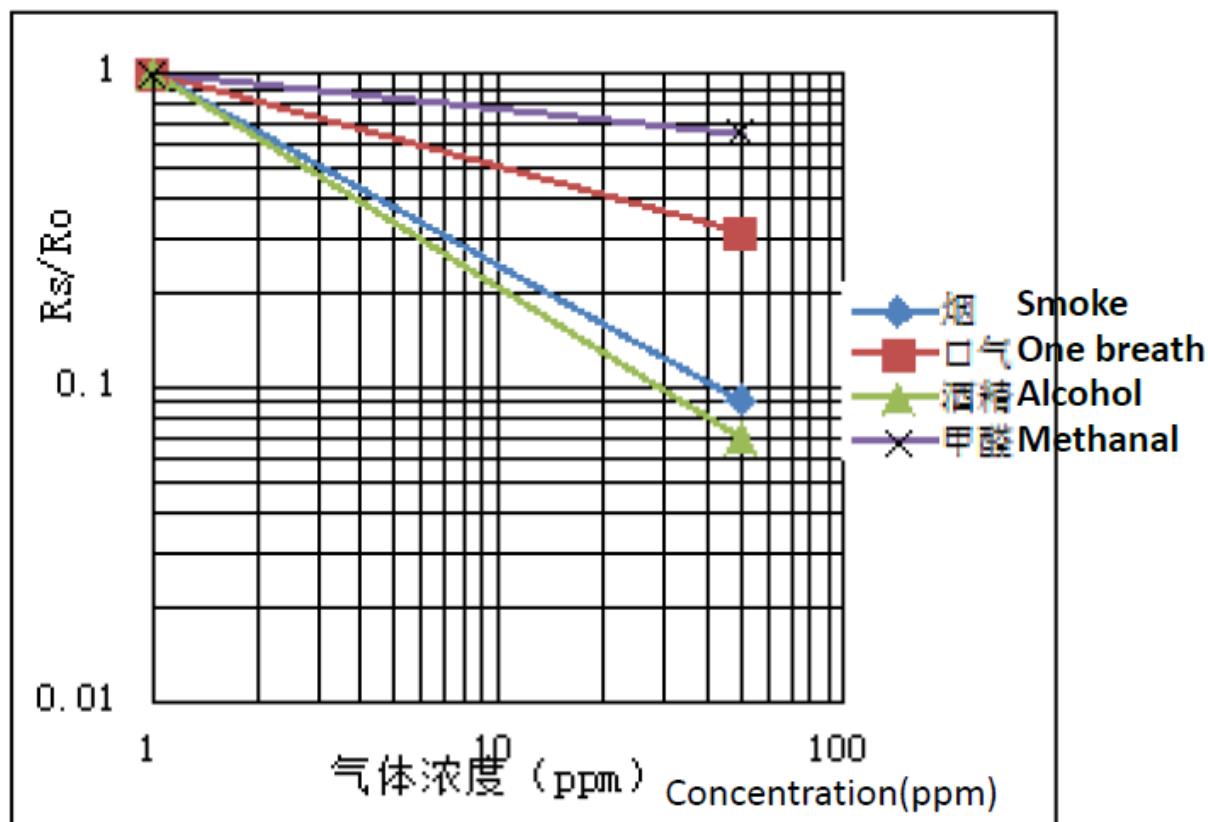
```
22.87 Celsius
53.16%
22.87 Celsius
53.16%
22.85 Celsius
53.16%
22.87 Celsius
53.12%
22.85 Celsius
53.12%
22.87 Celsius
53.12%
22.85 Celsius
53.09%
22.85 Celsius
```

CAPTEUR DE QUALITE D'AIR :

Le capteur de qualité d'air est branché en analogique à la carte Arduino. On récupère les informations du "Air QualitySensor" avec l'aide d'une bibliothèque ajoutée à notre programme. Ce dernier nous renvoie 0 (haute pollution), 1,2 ou 3(air pur).

Montage du système



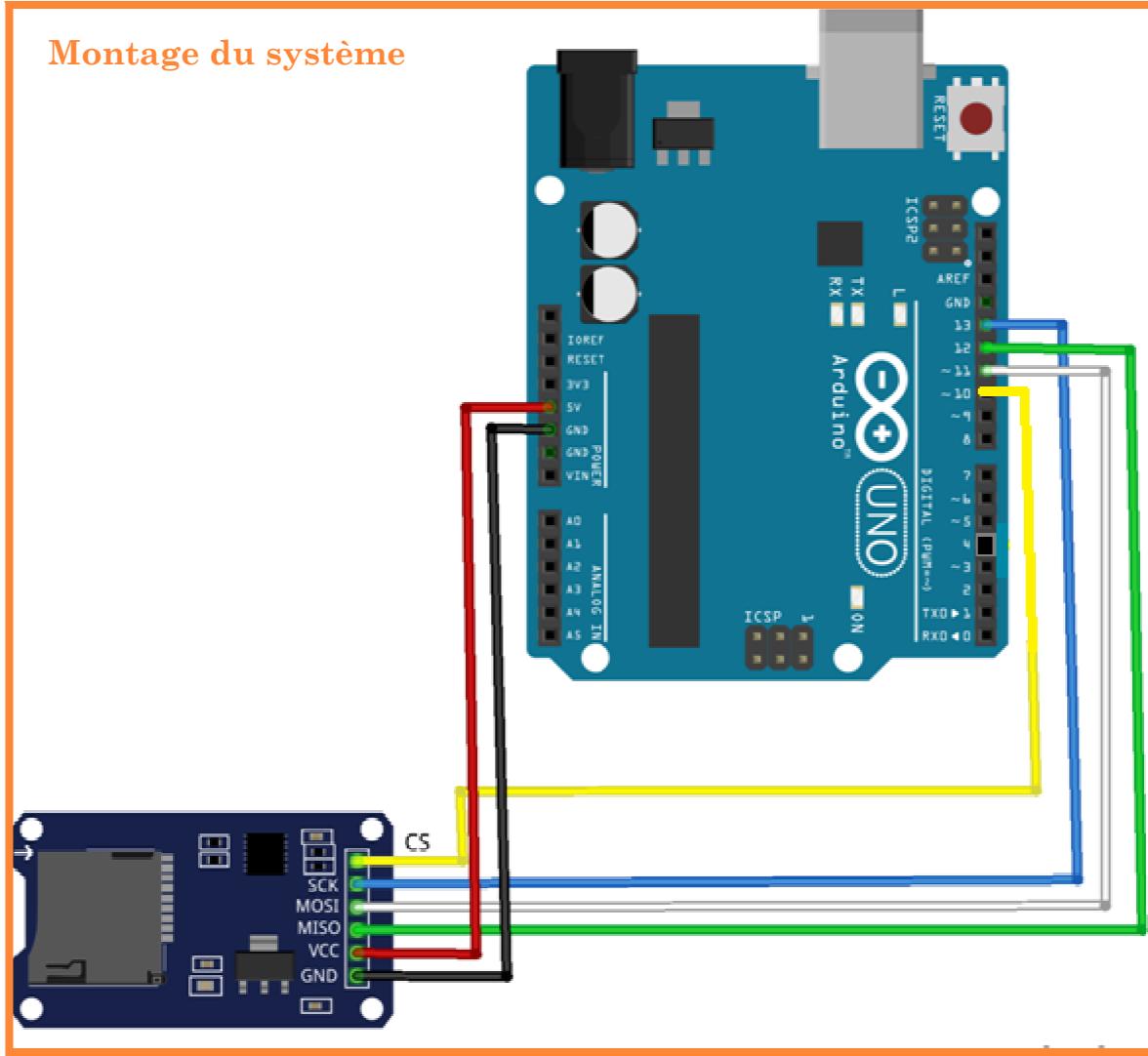


R_s est une résistance, dans un gaz ciblé avec une concentration différente, R_0 est la résistance du capteur à l'air propre. Toutes les valeurs sont déterminées dans des conditions de test standard.

CARTE SD :

La carte SD nous servira à sauvegarder les données du compteur d'énergie. Pour cela il m'a fallu apprendre à lire et à écrire dans une carte SD via une carte Arduino. Un article de **HACKABLE MAGAZINE**a fait un tutoriel à ce sujet, j'ai donc suivile tutoriel afin de réaliser mon objectif. Il fallait aussi relier la carte SD à la carte Arduino, nous avons donc utilisé un "MicroSDcard Adapter".

Montage du système



```

#include <SPI.h>
#include <SdFat.h>
#define SDCS 10
#define BUFFER_SIZE 100
SdFat sd;
uint8_t buf[BUFFER_SIZE];

void setup() {
    String bonjour="bonjour !";
    int n=0;
    String myString="";
    SdFile fichier;
    Serial.begin(9600);

    //init
    Serial.begin(9600);
    Serial.println("init SD");
    if(!sd.begin(SDCS,SPI_HALF_SPEED)){
        Serial.println("erreur init");
        return;
    }

    //ecriture
    if(!fichier.open(&sd, "toto.txt", O_RDWR|O_TRUNC|O_AT_END|O_SYNC)){
        Serial.println("Erreur");
        return;
    }
    fichier.println(bonjour);
    fichier.close();

    //liste du contenu de la carte
    sd.ls("/", LS_SIZE|LS_R);
    //lecture
    if(!fichier.open(&sd, "toto.txt", O_READ)){
        Serial.println("erreur");
        return;
    }

    //lecture nb d'octets dans le fichier
    n = fichier.read(buf, sizeof(buf));
    Serial.print(n);
    Serial.println(" octet(s) lu(s)");

    //lecture de la chaîne de caractère
    myString = String ((char *)buf);
    myString.trim();
    Serial.print("\\\"");
    Serial.print(myString);
    Serial.println("\\\"");

```

Valeur obtenue

```

init SD
    811 nom.txt
    765 FICHIER.TXT
    11 toto.txt
11 octet(s) lu(s)
"bonjour !"

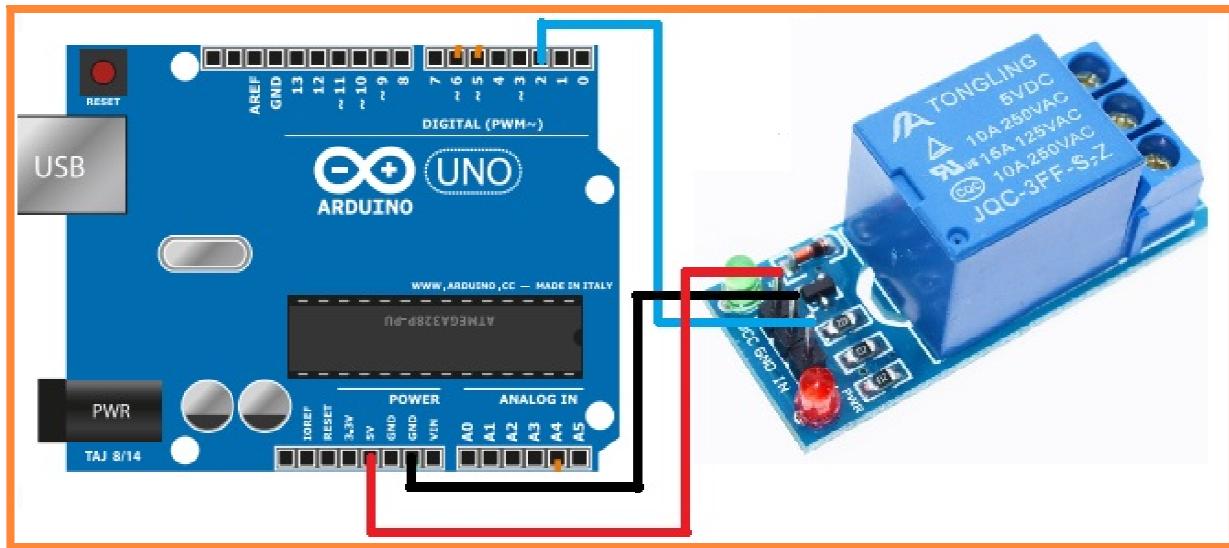
```

Code de test unitaire

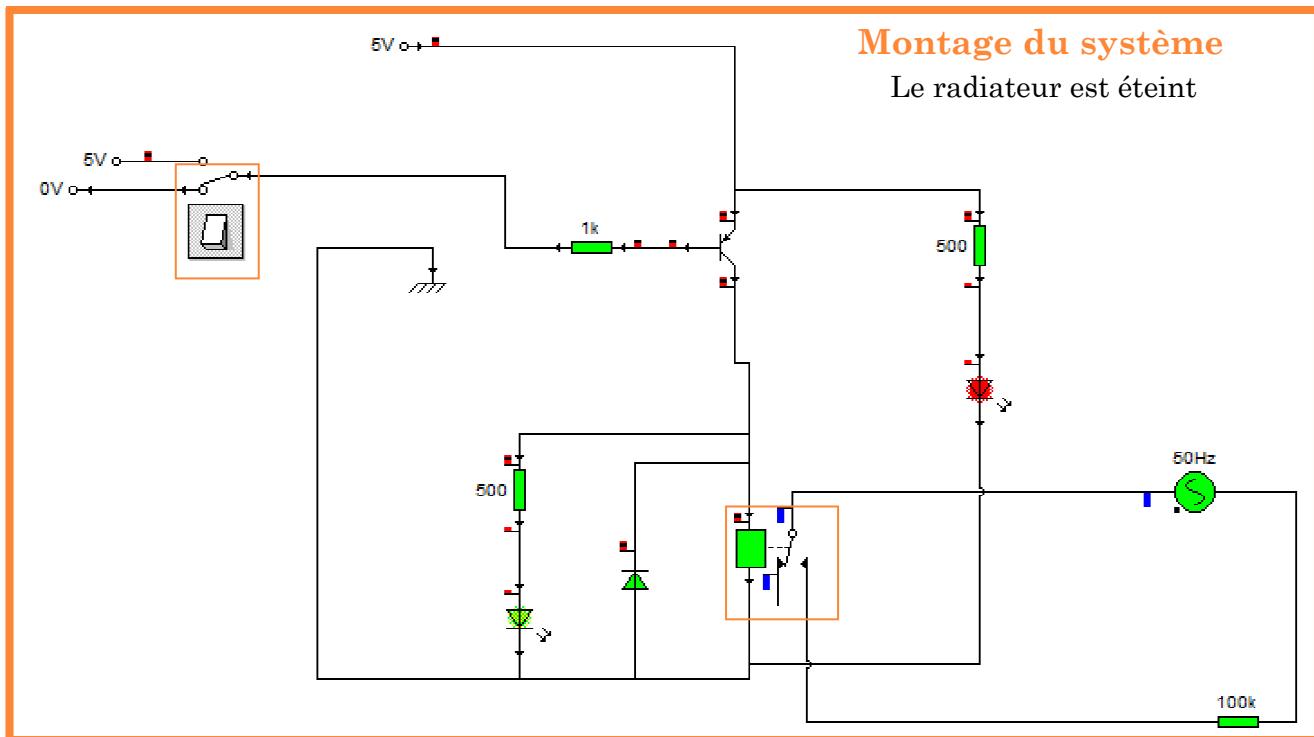
11 toto.txt : 11 correspond au nombre d'octet dans le fichier nommé **toto.txt**

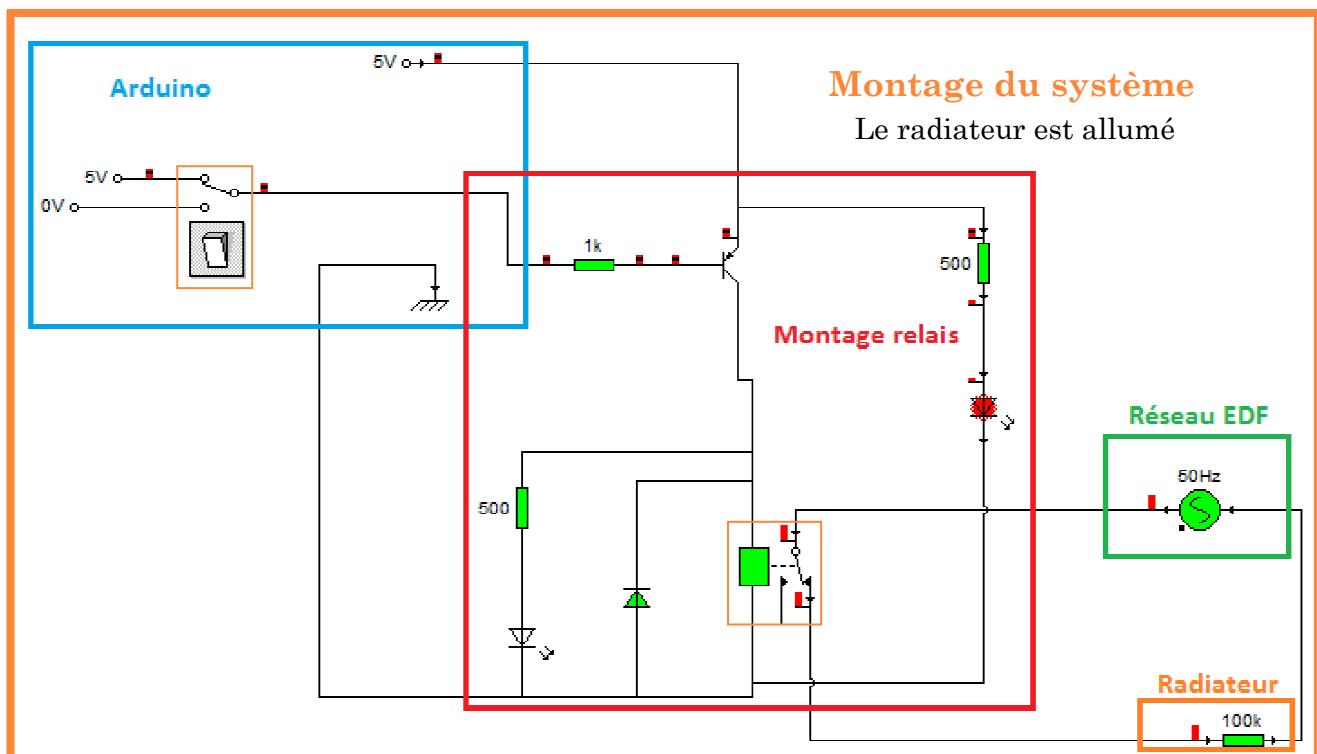
LE RADIATEUR :

Pour contrôler le radiateur avec une carte Arduino, il nous faut un relais qui nous permettre de contrôler avec le 5V de la carte, le radiateur qui est alimenté en 230V sur le réseau EDF.



Le schéma ci-dessus nous montre comment est reliée la carte aux CMS, carte qui permet de contrôler le relais.





```
#include "Arduino.h"
int pin_relay = 2;
char valeur;

void setup() {
    pinMode(pin_relay, OUTPUT);
    digitalWrite(pin_relay,HIGH);
    Serial.begin(9600);
    Serial.println("0 = allumer radiateur");
    Serial.println("1 = eteindre radiateur");
}

void loop() {

    if (Serial.available() > 0) {
        valeur = Serial.read();
        Serial.print("Radiateur = ");
        if (valeur=='1'){
            digitalWrite(pin_relay,HIGH);
            Serial.println("etteind");
        }
        else{
            digitalWrite(pin_relay,LOW);
            Serial.println("allumer");
        }
    }
}
```

Code de test unitaire

Valeur sur l'interface

0	Envoyer
0 = allumer radiateur	
1 = eteindre radiateur	
Radiateur = allumer	
Radiateur = etteind	
Radiateur = allumer	
Radiateur = etteind	
Radiateur = allumer	
Radiateur = etteind	
Radiateur = allumer	

LE MODULE RTC :

Le module RTC est un module qui permet de gérer l'heure, même quand le système n'est pas sous tension grâce à une petite pile. Pour le faire fonctionner j'ai fait appel à une bibliothèque sur GitHub. Avec les méthodes j'ai donc créé un mini programme donnant l'heure qui nous servira pour l'horodatage de données de consommation.

Montage du système

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

RTC_DS1307 rtc;

char daysOfTheWeek[7][12] = {"Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi", "Samedi", "Dimanche"};

void setup() {
    Serial.begin(9600);
    rtc.begin();
    rtc.isrunning();
}

void loop() {
    DateTime now = rtc.now();

    //affichage de la date
    Serial.print(now.year(), DEC);
    Serial.print('/');
    Serial.print(now.month(), DEC);
    Serial.print('/');
    Serial.print(now.day(), DEC);
    Serial.print(" (");

    Serial.print(daysOfTheWeek[now.dayOfTheWeek()]);
    Serial.print(" ");

    //affichage de l'heure
    Serial.print(now.hour(), DEC);
    Serial.print(":");
    Serial.print(now.minute(), DEC);
    Serial.print(":");
    Serial.print(now.second(), DEC);
    Serial.println();
}

delay(1000);
}
```

Valeur obtenue

2018/2/12 (Lundi) 11:39:23
2018/2/12 (Lundi) 11:39:24
2018/2/12 (Lundi) 11:39:25
2018/2/12 (Lundi) 11:39:26
2018/2/12 (Lundi) 11:39:28
2018/2/12 (Lundi) 11:39:29
2018/2/12 (Lundi) 11:39:30
2018/2/12 (Lundi) 11:39:31
2018/2/12 (Lundi) 11:39:32
2018/2/12 (Lundi) 11:39:33
2018/2/12 (Lundi) 11:39:34
2018/2/12 (Lundi) 11:39:35
2018/2/12 (Lundi) 11:39:36
2018/2/12 (Lundi) 11:39:37
2018/2/12 (Lundi) 11:39:38

Code de test unitaire

COMPTEUR ELECTRIQUE :

Avec le compteur électrique nous allons pouvoir récupérer le nombre de KWh que le radiateur va consommer. Il délivrera une impulsion tout les 100Wh que nous allons récupérer avec la carte Arduino.

J'ai pour cela utilisé la pin 2 de la carte car elle est la seule pin avec la numéro 3 à posséder une interruption externe. Ensuite nous avons utilisé la résistance de pull-up interne de la pin 2 afin de ne pas avoir à monter le composant sur une carte.

```
int interruptePinCompteur = 2;

void setup() {
    Serial.begin(9600);
    //INPUT_PULLUP nous sers a utiliser la résistance de pull_up lier à la broche
    pinMode(interruptePinCompteur, INPUT_PULLUP);
    pinMode(3, OUTPUT);
    //déclaration d'une interruption le premier paramètre est la broche a utiliser
    //le deuxième paramètre est est le nom de la fonction
    //le troisième paramètre est l'état de la pate qui déclenchera l'interruption.
    attachInterrupt(digitalPinToInterrupt(interruptePinCompteur), interruption, LOW);
}

void loop() {

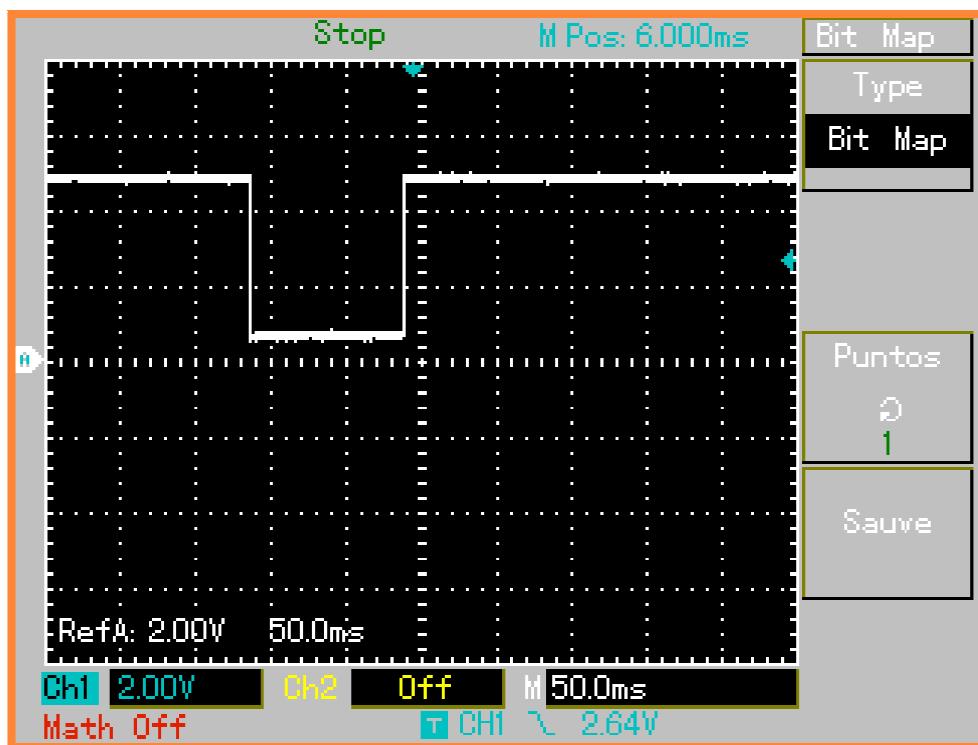
}

void interruption(){}
    //le delay nous permet de ne pas lancer l'interruption
    //en cas de perturbation électrique
    delay(50);
    //on vérifie l'état de la broche
    if (digitalRead(interruptePinCompteur)==0) {
        Serial.println("bonjour");
        //on attend que la broche repasse a l'état haut
        while (digitalRead(interruptePinCompteur)!=1);
    }
}
```

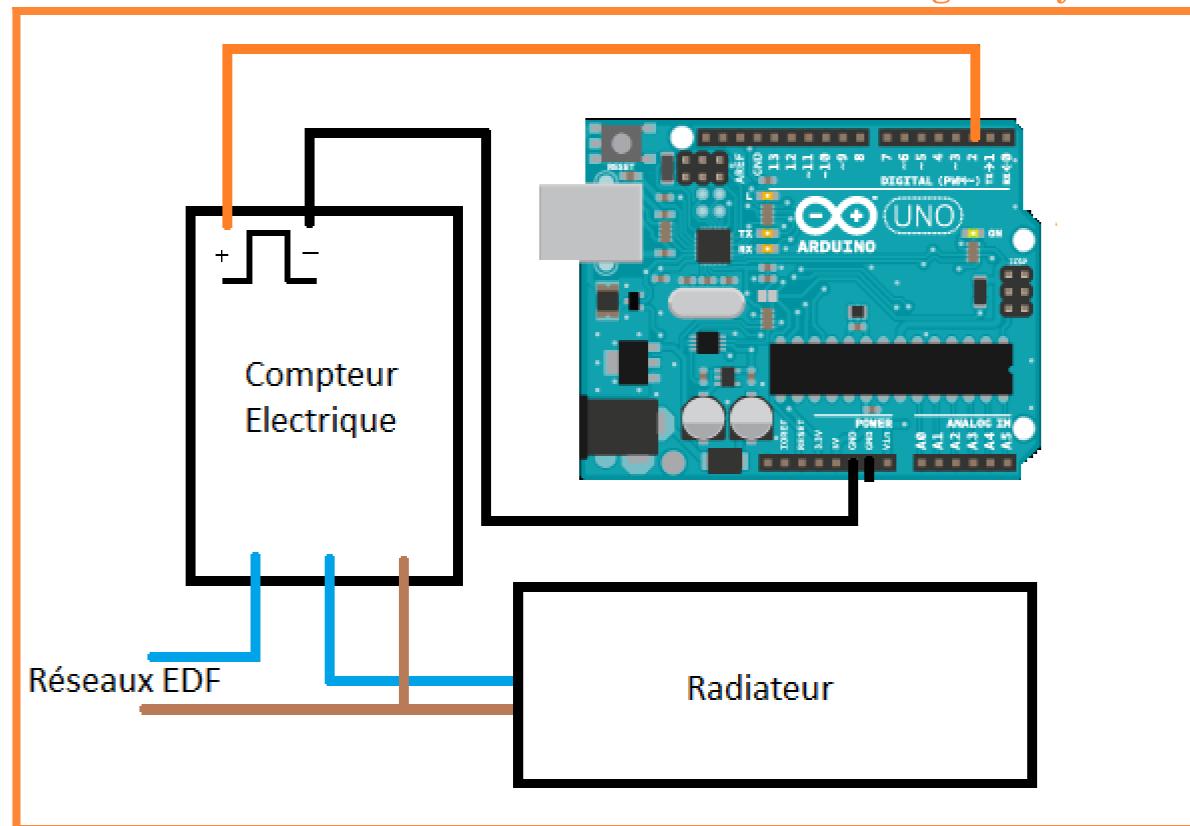
Code de test unitaire

Le délai est de 50 ms, ce qui correspond à la moitié de la durée de l'impulsion générée par le compteur. Ensuite, on revérifie l'état de la broche. Cette opération nous permet d'éliminer les parasites liés aux perturbations électriques.

Voici à l'oscilloscope, l'impulsion générée par le compteur électrique :



Montage du système

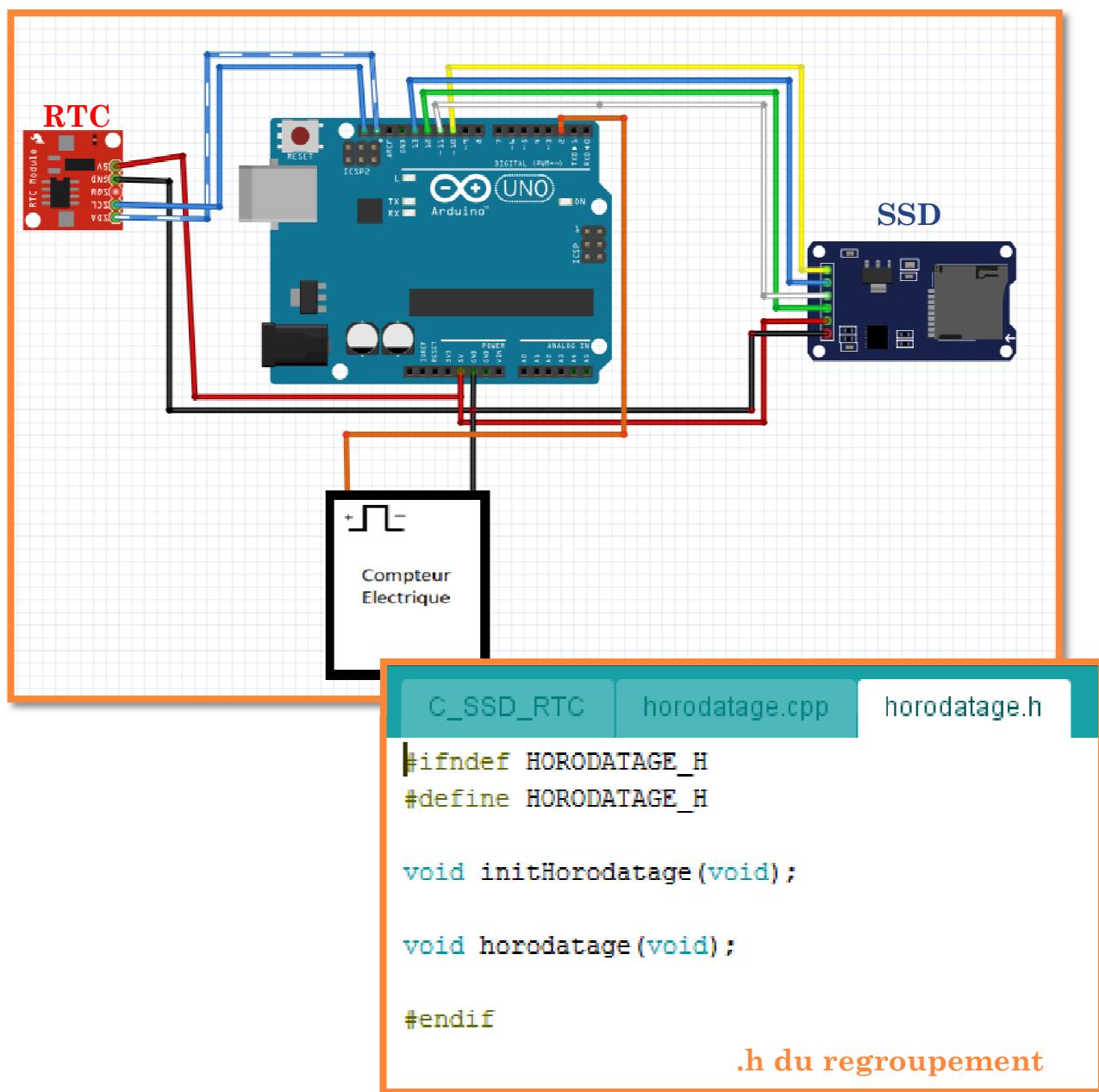


3 - REGROUPEMENT DES COMPOSANTS

Maintenant que tous les tests unitaires sont faits, nous pouvons regrouper les composants sur une seule carte et les codes dans un seul fichier.

LA CARTE SSD, LE MODULE RTC ET LE COMPTEUR ÉLECTRIQUE :

Montage du système



C_SSD_RTC horodatage.cpp horodatage.h

```
#include "horodatage.h"

int interruptePinCompteur = 3; //Variable Compteur
unsigned short flagCompteurEnergie=0;

void setup() {
    Serial.begin(9600);

    //Setup Compteur interruption
    pinMode(interruptePinCompteur, INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(interruptePinCompteur), interruption, LOW);

    initHorodatage();
}

void loop() {

    if (flagCompteurEnergie==1){
        horodatage();
        flagCompteurEnergie=0;
    }
}

void interruption(){
    delay(50);
    if (digitalRead(interruptePinCompteur)==0) {
        flagCompteurEnergie=1;
        while(digitalRead(interruptePinCompteur)!=1);
    }
}
```

Code principale du regroupement.

C_SSD_RTC horodatage.cpp horodatage.h

```
#include <SPI.h>          //
#include <SdFat.h>          //Bibliothèque carte SD
#include <RTClib.h>          //Bibliothèque RTC
#include "horodatage.h"
#define SDCS 10              //
#define BUFFER_SIZE 250       //Define SD

SdFat sd;                  //
uint8_t buf[BUFFER_SIZE];  //Variable SD
float pulsion = 0.1;
RTC_DS1307 rtc;            //Variable RTC

void initHorodatage() {
    //Setup SD
    Serial.println("init SD");
    if(!sd.begin(SDCS,SPI_HALF_SPEED)){
        Serial.println("erreur init");
        return;
    }
    rtc.begin();
    rtc.isrunning();
}
```

L'un des avantages d'exécuter le programme d'interruption dans le loop est que nous évitons les conflits d'interruption, car la bibliothèque qui nous sert à horodater utilise aussi une interruption, donc si nous avions exécuté la fonction horodatage dans le programme d'interruption il y aurait eu conflit d'interruption (une interruption dans une interruption).

```

void horodatage() {
    SdFile fichier;

    //écriture dans le fichier txt Compteur_Elec dans la SD
    if(!fichier.open(&sd, "Compteur_Elec_Energie.txt", O_RDWR|O_TRUNC|O_AT_END|O_SYNC)) {
        Serial.println("Erreur");
        return;
    }
    fichier.print(pulsion);
    fichier.print("KWh ");
    DateTime now = rtc.now();
    //affichage de la date
    fichier.print(now.year(), DEC);
    fichier.print('/');
    fichier.print(now.month(), DEC);
    fichier.print('/');
    fichier.print(now.day(), DEC);
    fichier.print(" ");
    //affichage de l'heure
    fichier.print(now.hour(), DEC);
    fichier.print(":");
    fichier.print(now.minute(), DEC);
    fichier.print(":");
    fichier.println(now.second(), DEC);
    fichier.close();

    sd.ls("/", LS_SIZE|LS_R);

    //lecture du contenu du fichier txt Compteur_Elec dans la SD
    if(!fichier.open(&sd, "Compteur_Elec_Energie.txt", O_READ)) {
        Serial.println("erreur");
        return;
    }
    fichier.read(buf, sizeof(buf));
    String myString = String ((char *)buf);
    myString.trim();
    Serial.print("\n");
    Serial.print(myString);
    Serial.println("\n");
    fichier.close();
    pulsion=pulsion+0.1;
}

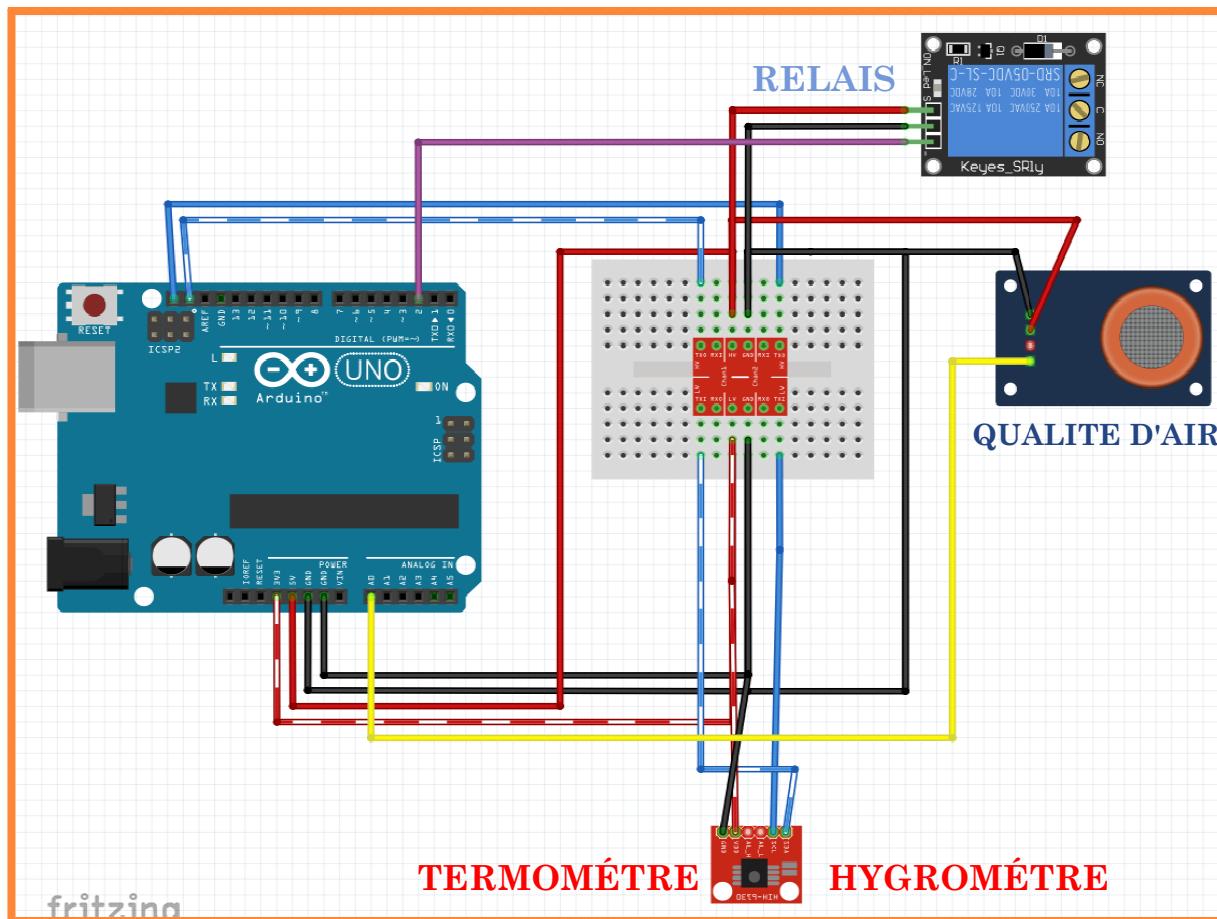
```

.cpp partie 2 du regroupement

Dans cette partie nous avons regroupé tout ce qui concerne l'horodatage de la consommation électrique. Le compteur crée une impulsion à laquelle on ajoute une date, une heure et une valeur en Watt/h, puis cette trame est sauvegardée sur une carte SD dans un fichier texte (.txt).

LE THERMOMETRE, L'HYGROMETRE, LE CAPTEUR DE QUALITE D'AIR ET LE RELAIS :

Montage du système



```

T_H_AQ_R    capteur.cpp    capteur.h    relai.cpp    relai.h

```

```

#include "capteur.h"
#include "relai.h"

void setup() {
  Serial.begin(9600);
  initRelai();
  initCapteur();
}

void loop() {
  temperature();
  humidite();
  qualiteAir();
  relai();

  Serial.println("");
  delay(2000);
}

```

Code principale du regroupement.

T_H_AQ_R capteur.cpp capteur.h relai.cpp relai.h

```
#include "capteur.h"
#include <HIH6130.h>
#include<AirQuality.h>
```

```
HIH6130 rht(0x27);
```

```
AirQuality airqualitysensor;
int current_quality =-1;
```

```
void initCapteur() {
    Serial.println("initialisation des capteurs...");
    rht.begin();
    airqualitysensor.init(A0);
    delay (5000);
}
```

```
int temperature() {
    rht.readRHT();
    Serial.print(rht.temperature());
    Serial.println (" Celsius");
}
```

```
int humidite() {
    rht.readRHT();
    Serial.print(rht.humidity());
    Serial.println (" %");
}
```

```
int qualiteAir() {
    current_quality=airqualitysensor.slope();
    if (current_quality >= 0)// if a valid data returned.
    {
        if (current_quality==0)
            Serial.println("Pollution importante !!");
        else if (current_quality==1)
            Serial.println("Pollution");
        else if (current_quality==2)
            Serial.println("Faible pollution");
        else if (current_quality ==3)
            Serial.println("Air pure");
    }
}
```

capteur.cpp partie 1 du regroupement

```
ISR(TIMER1_OVF_vect) //timer
{
    if(airqualitysensor.counter==61)//une fois que le nombre de marqueur
        //atteind 61 (~30ms) on exécute les
        //lignes de commande suivante
    {
        /*On récupère la dernière valeur avant de la remplacer par
        *une valeur plus récentes puis on remet le nombre de marqueur
        * à zéro et on relance le timer*/
        airqualitysensor.last_vol=airqualitysensor.first_vol;
        airqualitysensor.first_vol=analogRead(A0);
        airqualitysensor.counter=0;
        airqualitysensor.timer_index=1;
        PORTB=PORTB^0x20;
        /*La bibliothèque compare les deux valeur last_vol et
        *first_vol pour nous indiquer la qualité de l'air*/
    }
    else
    {
        airqualitysensor.counter++;
    }
}
```

capteur.cpp partie 2 du regroupement

T_H_AQ_R	capteur.cpp	capteur.h	relai.cpp	relai.h
----------	-------------	-----------	-----------	---------

```
#ifndef CAPTEUR_H
#define CAPTEUR_H

int temperature (void);

int humidite (void);

void initCapteur (void);

int qualiteAir (void);

#endif
```

Capteur.h du regroupement

T_H_AQ_R	capteur.cpp	capteur.h	relai.cpp	relai.h
----------	-------------	-----------	-----------	---------

```

#include "relai.h"
#include "Arduino.h"
int pin_relay = 2;
char valeur;

void initrelai(){
    pinMode(pin_relay, OUTPUT);
    digitalWrite(pin_relay,HIGH);
    Serial.println("entre 0 pour allumer le radiateur");
    Serial.println("entre 1 pour eteindre le radiateur\n");
}

void relai(){
    if (Serial.available() > 0) {
        valeur = Serial.read();
        Serial.print("Radiateur = ");
        if (valeur=='1'){
            digitalWrite(pin_relay,HIGH);
            Serial.println("eteind");
        }
        else{
            digitalWrite(pin_relay,LOW);
            Serial.println("allumer");
        }
    }
}

```

capteur.cpp du regroupement

T_H_AQ_R	capteur.cpp	capteur.h	relai.cpp	relai.h
----------	-------------	-----------	-----------	---------

```

#ifndef RELAI_H
#define RELAI_H

void initrelai(void);

void relai (void);

#endif

```

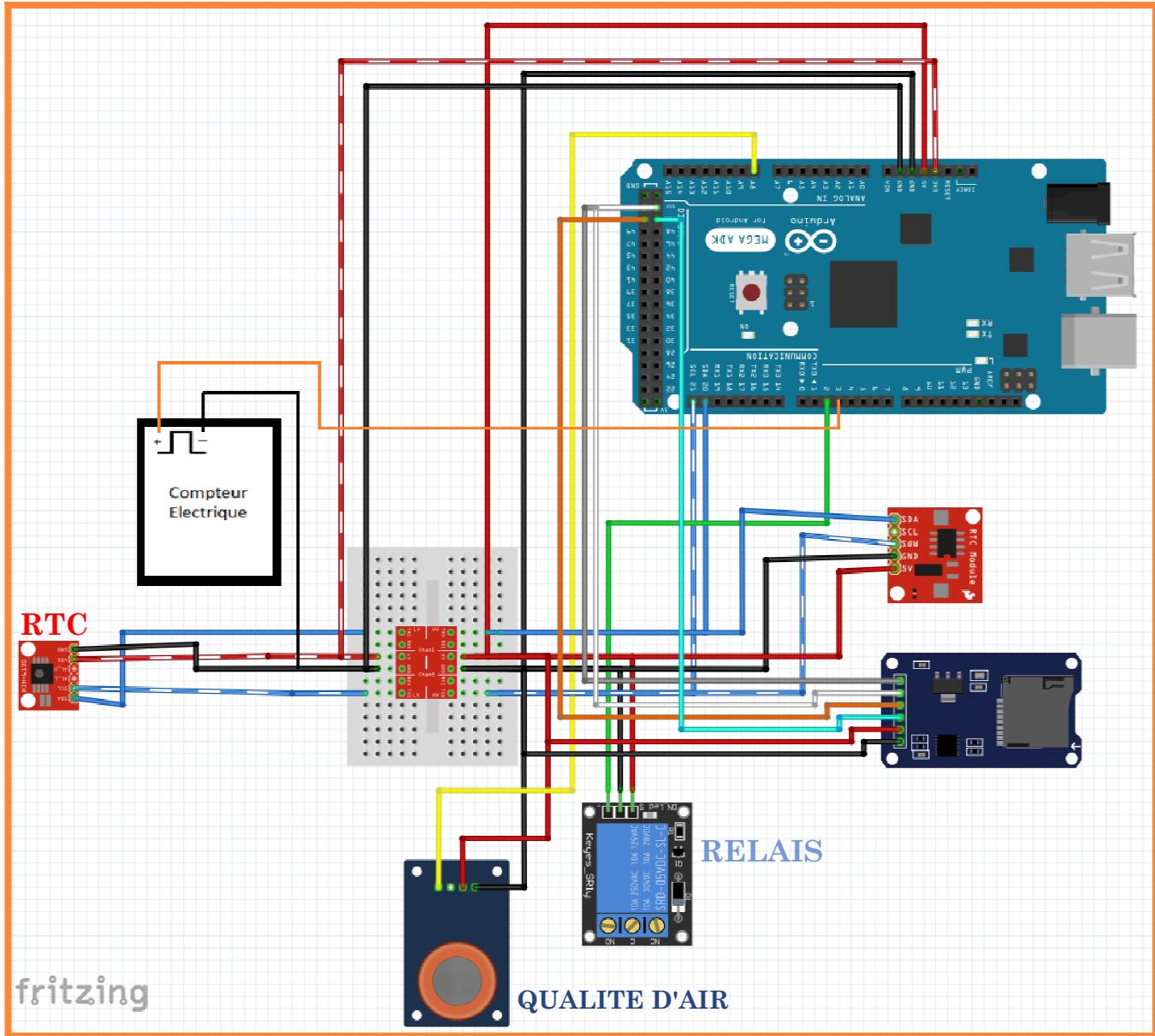
Capteur.h du regroupement

Ici, nous avons regroupé les capteurs et actionneurs de la partie confort. On y récupère la température(°C), le taux d'humidité (%), la qualité de l'air et on contrôle le relais qui agira sur le radiateur.

REGROUPEMENT FINAL DES COMPOSANTS :

Une fois la première partie de regroupement réalisée, nous pouvons à présent regrouper la totalité de cette partie en un seul montage et un seul programme. Nous pourrons ensuite regrouper les codes de la partie de l'étudiant 1 avec notre partie.

Montage du système



Lors du branchement du module SD nous avons changé de **pin** pour la communication SPI car elle varie en fonction de la carte que nous utilisons.

Arduino UNO : MOSI = **11** / MISO = **12** / SCK = **13** / SS = **10**

Arduino MEGA : MOSI = **51** / MISO = **50** / SCK = **52** / SS = **53**

```
//config timer2
TCCR2A=0x00;
TCCR2B|=(1<<CS22)|(1<<CS21)|(1<<CS20);
TIMSK2|=(1<<TOIE2);
sei();
```

Ici nous paramétrons un timer qui nous permettra de récupérer les valeurs et états de nos capteurs et actionneurs.

TCCR2A nous indique que le timer est en mode normal.

TCCR2B est le coefficient diviseur de la fréquence du CPU.

TIMSK2 nous indique pour que le timer entre en interruption à chaque débordement (Overflow).

sei() : Active les interruptions.

Ici notre timer entre en interruption 61 fois par seconde, ce qui correspond à la levée des drapeaux. Le premier drapeau se lève tous les 305 débordements du timer, donc toutes les 5 secondes. Quant au deuxième drapeau, il se lève toutes les 10 secondes.

```
ISR(TIMER2_OVF_vect) {
    cpt++;
    if (cpt==305) {
        flagCompteurTimer=1;
    }
    if (cpt==610) {
        flagCompteurTimer=2;
    }
}
```

```
if (flagCompteurTimer==1){
    maison.temperature=temperature();
    maison.humidite=humidite();
    maison.etatRadiateur=etatRelai();
    flagCompteurTimer=0;
}
if (flagCompteurTimer==2){
    maison.qualiteAir=qualiteAir();
    maison.temperature=temperature();
    maison.humidite=humidite();
    maison.etatRadiateur=etatRelai();
    flagCompteurTimer=0;
    cpt=0;
}
```

Nous voyons maintenant que les données sont stockées dans l'objet maison toutes les 5 et 10 secondes.

Les drapeaux sont remis à zéro ainsi que le compteur, à la fin du programme, lancé par le deuxième drapeau.

```

while(Serial2.available() > 0) {
    double c = Serial2.read();
    if (c==8){
        maison.radiateurMode=true;
    }
    if (c==9){
        maison.radiateurMode=false;
    }
    if (c>=10 && c<=30){
        maison.temperatureUtilisateur = c;
    }
    if (c==6){
        if (maison.radiateur==true){
            maison.radiateur=false;
        }
    }
    if (c==7){
        if (maison.radiateur==false){
            maison.radiateur=true;
        }
    }
}
}

```

Ce morceau du code nous permet de récupérer les données envoyées par la tablette. Ces données sont des valeurs qui correspondent à une requête soumise par l'utilisateur. Par exemple si la valeur envoyée est 8, nous passons en mode manuel, si la valeur est 9 nous passons en mode automatique. De cette manière nous pouvons contrôler le mode de fonctionnement du radiateur.

Maison.temperatureUtilisateur prend la valeur choisie par l'utilisateur pour son chauffage. Ainsi, lorsque le chauffage sera en mode automatique, le chauffage restera à la température souhaitée.

```

void relai(bool mode, double temperature, double temperatureUtilisateur, bool valeur) {
    if (mode==false){
        if (valeur==true){
            digitalWrite(pin_relay,HIGH);
        }
        if (valeur==false){
            digitalWrite(pin_relay,LOW);
        }
    }
    if (mode==true){
        if(temperature<=temperatureUtilisateur-1){
            digitalWrite(pin_relay,HIGH);
        }
        if(temperature>=temperatureUtilisateur){
            digitalWrite(pin_relay,LOW);
        }
    }
}

```

Ici nous avons le code qui permet de contrôler le chauffage en automatique et en manuel. On peut remarquer qu'il y a une plage d'hystéresis de 1 degré qui nous permet de ne pas casser notre relais.

```
#ifndef __MAISON_H__
#define __MAISON_H__
#include "Arduino.h"
class Maison
{
public:
    bool lumiere=false;
    bool volet1=false;
    bool volet2=false;
    bool incendie=false;
    bool mouvement=false;
    float luminosite;

    double temperatureUtilisateur=21;
    bool etatRadiateur;
    bool radiateur=false;
    double temperature;
    char qualiteAir;
    double humidite;
    float consommation;
    bool radiateurMode=false;

    int annee;
    int jour;
    int mois;
    int heure;
    int minutes;
    int seconde;

private:
};

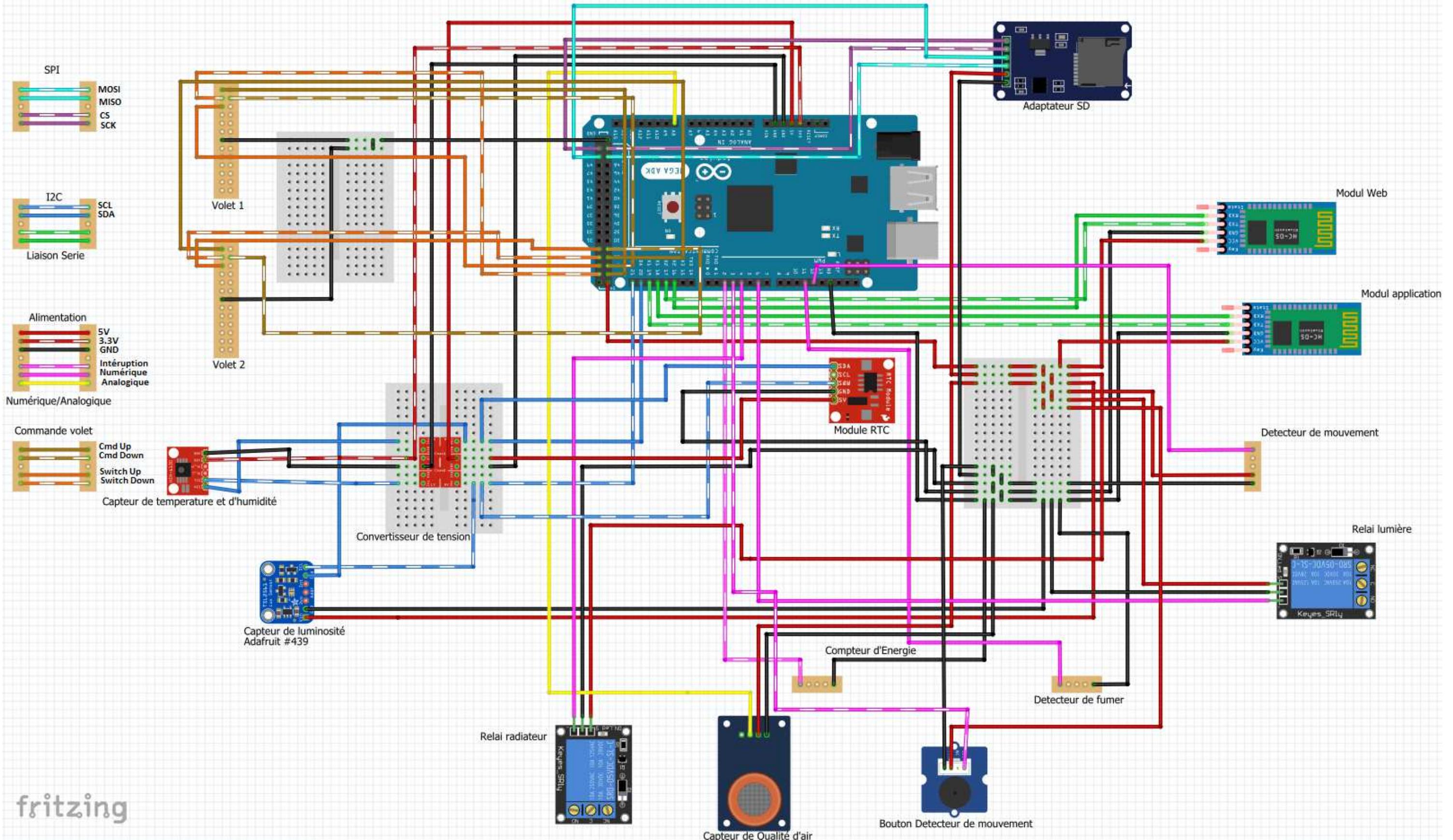
#endif
```

Voici la première version de l'objetmaison. Il sera amélioré par la suite avec l'arrivée du code de l'étudiant 1.

Dans cet objet nous avons toutes les variables liées à un capteur ou à un actionneur (sonétat, les valeurs qu'il renvoie, son mode de fonctionnement, etc.).

L'heure et la date sont aussi sauvegardées séparément pour pouvoir par la suite les envoyer un par un à la tablette ou au serveur Web.

4 - REGROUPEMENT DES DIFFÉRENTES PARTIES.



Dans cette partie, nous verrons le travail que nous avons réalisé lors du regroupement de toutes nos parties. Dans un premier temps, nous avons réorganisé le code, puis nous avons créé de nouvelles bibliothèques plus intuitives et plus claires.

Ces nouvelles bibliothèques ont leur propre fonction initialisation, ce qui rend le `void setup()` beaucoup plus intuitif.

```
void setup() {
    initSerial();
    initTimer2();
    initActionneur();
    initHorodatageConsomation();
    initCapteur();

    //Setup Compteur d'energie interruption
    pinMode(interruptePinCompteur, INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(interruptePinCompteur), interruptionCompteur, LOW);

    //setup bouton alarme interruption
    pinMode(interruptBouton, INPUT);
    attachInterrupt(digitalPinToInterrupt(interruptBouton), interruptionBouton, RISING);
}
```

```
void initCapteur(){
    Serial.println("initialisation des capteurs...");
    rht.begin();
    pinMode(CAPTEUR_INCENDIE, INPUT);
    pinMode(CAPTEUR_MOUVEMENT, INPUT);
    airqualitysensor.init(A0);
    delay(5000);
    Serial.println("Capteurs initialisés.");
}
```

```
void initTimer2(){
    //config timer2
    TCCR2A=0x00;
    TCCR2B=(1<<CS22)|(1<<CS21)|(1<<CS20);
    TIMSK2=(1<<TOIE2);
    sei();
}
```

```
void initSerial(){
    Serial.begin(9600);
    Serial1.begin(9600);
    Serial2.begin(9600);
    Serial3.begin(9600);
}
```

```
#include "capteur.h"
#include "controleActionneur.h"
#include "horodatageConsomation.h"
#include "gestionMaison.h"
```

```
void initActionneur(){
    pinMode(pinThermostat, OUTPUT);
    pinMode(pinLumiere, OUTPUT);

    pinMode(cmdUp, OUTPUT);
    pinMode(cmdDown, OUTPUT);
    pinMode(switchUp, INPUT);
    pinMode(switchDown, INPUT);

    pinMode(cmdUp2, OUTPUT);
    pinMode(cmdDown2, OUTPUT);
    pinMode(switchUp2, INPUT);
    pinMode(switchDown2, INPUT);
}
```

```
void initHorodatageConsomation(){
    //Setup SD
    Serial.println("init SD");
    if(!sd.begin()){
        Serial.println("erreur init");
        return;
    }

    rtc.begin();
    rtc.isrunning();
}
```

LA BIBLIOTHÈQUE CAPTEUR.H :

Dans cette bibliothèque, nous gérons tous les capteurs (humidité, luminosité, température, qualité d'air, mouvement et incendie), ainsi que l'envoi de SMS créé par l'étudiant 1.

```
#include "capteur.h"
#include "Arduino.h"
#include <HIH6130.h>
#include <AirQuality.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_TSL2561_U.h>
#define CAPTEUR_INCENDIE 35
#define CAPTEUR_MOUVEMENT 36

Adafruit_TSL2561_Unified tsl = Adafruit_TSL2561_Unified(TSL2561_ADDR_FLOAT, 12345);
|
HIH6130 rht(0x27);

AirQuality airqualitysensor;
int current_quality = -1;

int answer;
char aux_string[30];
char phone_number[]{"+33648977501"};

void initCapteur(){
    Serial.println("initialisation des capteurs...");
    rht.begin();
    pinMode(CAPTEUR_INCENDIE, INPUT);
    pinMode(CAPTEUR_MOUVEMENT, INPUT);
    airqualitysensor.init(A0);
    delay (5000);
    Serial.println("Capteurs initialisés.");
}
```

Chaque capteur renvoie une valeur qui est récupérée dans l'objet maison.

```
maison.qualiteAir=capteurQualiteAir();
maison.temperature=capteurTemperature();
maison.humidite=capteurHumidite();
maison.etatRadiateur=etatThermostat();
maison.luminosite=capteurLuminosite();
```

```
if (maison.incendie == false){
    maison.incendie=capteurIncendie();
}

if (maison.alarme==true){
    if (maison.mouvement == false){
        maison.mouvement=capteurMouvement();
```

LA BIBLIOTHÈQUE GESTIONMAISON.H :

```

class Maison
{
public:
    bool lumiere=false;
    bool volet1=false;
    bool volet2=false;
    float luminosite;
    bool alarme=false;
    bool incendie;
    bool mouvement;

    double temperatureUtilisateur=21;
    bool radiateur=false;
    double temperature;
    int qualiteAir;
    double humidite;
    float consomation;

    bool etatRadiateur;
    bool lumiereEstat=false; //_false=0=etteind|true=1=allumer
    bool volet1Estat=false; //_false=0=etteind|true=1=allumer
    bool volet2Estat=false; //_false=0=etteind|true=1=allumer

    bool voletMode=true; //_true=0=mode manuel|false=1=mode automatique
    bool radiateurMode=false; //_false=0=mode manuel|true=1=mode automatique

    int annee;
    int jour;
    int mois;
    int heure;
    int minutes;
    int seconde;

    void emissionTrame (void);

    void lectureTablette (void);

private:
};

```

Voici l'objet maison finalisé, on y stocke toute valeur utilisable ainsi que deux méthodes dont nous allons nous servir pour communiquer avec la tablette et pour envoyer nos données au serveur web.

Ceci est une première ébauche de notre objet, il est nettement améliorable, surtout au niveau des règles d'encapsulation.

FONCTION D'ENVOI DE LA TRAME :

```
void Maison::emissionTrame () {
    String trame;

    //tablette
    trame += lumiere;
    trame += "!";
    trame += luminosite;
    trame += "!";
    trame += volet1Etat;
    trame += "!";
    trame += volet2Etat;
    trame += "!";
    trame += etatRadiateur;
    trame += "!";
    trame += temperature;
    trame += "!";
    trame += consomation;
    trame += "!";
    trame += humidite;
    trame += "!";
    trame += qualiteAir;
    //relevé date
    trame += "!";
    trame += jour;
    trame += "!";
    trame += mois;
    trame += "!";
    trame += annee;
    trame += "!";
    trame += heure;
    trame += "!";
    trame += minutes;
    trame += "!";
    Serial.println(trame);
    Serial2.println(trame);
    Serial3.println(trame);
    trame = "";
}
```

Dans cette fonction nous ajoutons les valeurs récupérées des différents capteurs les unes après les autres, en les séparant par un point d'exclamation.

Cette opération nous génère une trame que nous allons envoyer.

État de la lumière 0 = éteint et 1 = allumé

Luminosité en LUX

État du volet n°1 0 = fermé et 1 = ouvert

État du volet n°1 0 = fermé et 1 = ouvert

État du radiateur 0 = éteint et 1 = allumé

Température en °C

Consommation en KwH

Taux d'humidité en %

Qualité de l'air de 0 à 3

Heure et date : jour ! mois ! année ! heure ! minutes !

CONCLUSION :

Mon ressenti :

Ce projet m'a beaucoup plu. Le travail d'équipe est quelque chose de très appréciable, et le fait d'être le chef de projet m'a beaucoup apporté, de la confiance en soi, des techniques d'organisation, et j'ai pris connaissance de mes capacités de travail.

Les évolutions possibles :

Pour les radiateurs il faudrait envisager de les piloter automatiquement avec une plage d'hystérésis plus fine, car 1°C de différence dans une maison se ressent considérablement.

Il faudrait aussi faire en sorte que les actionneurs puissent changer d'état en même temps et pas les uns après les autres.

Une amélioration de l'objet maison est aussi possible et recommandé, surtout au niveau des règles d'encapsulation.

