

Horloge avec station météo connectée

Auteur :
Grégory

Préambule

Afin d'afficher l'heure ainsi que d'autres informations à destination des usagers du Fablab de Janzé, nous avons mis en place une horloge connectée indiquant les horaires d'ouverture du Fablab, le numéro de téléphone et les conditions météo de la journée.

Matériel nécessaire

Pour la réalisation de ce projet, nous avons eu besoin d'une matrice LED 32x64, d'un DHT22, d'un Esp8266 et d'une alimentation 5V 4A.

Etape 1 : Branchement de la matrice LED et du DHT22

Etape 2 : Le script et les fichiers nécessaires

Pour la connexion météo, nous nous connecterons au serveur <https://www.openweathermap.org/>.

Il faudra créer un compte et utiliser l'API.

Pour la Librairie ArduinoJson, ATTENTION, nous avons utilisé la version 5.13.4, sinon, la déclaration des objets ne fonctionnera pas.

En téléchargement ici : <https://cloud.ccprf.fr/index.php/s/AMpDbdTqXGwJ4iY>

Voici le code découpé en différentes parties :

Nous retrouvons dans le code des impressions sur la console, mais elles peuvent être retirées. Elles servaient au suivi pendant la programmation.

```
/*  
 * Horloge avec NODEMCU et matrice LED RGB 32x64  
 */  
  
#include <SPI.h>  
  
#include <Wire.h>  
  
//Pour la connexion WIFI  
#include <ESP8266WiFi.h>  
  
const char* ssid = « Livebox-**** »;  
const char* password = « ***** »;  
  
//Pour la connexion NTP  
#include <WiFiUdp.h>  
  
#include <NTPClient.h>
```

```

//Creation objet WIFI UDP puis du client NTP
WiFiUDP ntpUDP;

// Avec intervalle de mise à jour de 60 sec ...

// ... et offset en secondes cad decalage horaire, pour GMT+1 mettre 3600, GMT +8 mettre 28800, etc.

// ... et le serveur NTP utilise est : europe.pool.ntp.org
NTPClient timeClient(ntpUDP, « europe.pool.ntp.org », 3600, 60000);

//Pour DHT22

#include « DHT.h » // Librairie des capteurs DHT

#define DHTPIN 0 // Changer le pin sur lequel est branché le DHT

#define DHTTYPE DHT22 // DHT 22 (AM2302)

DHT dht(DHTPIN, DHTTYPE);

//Pour météo serveur

#include <ArduinoJson.h>

// déclaration pour la connexion API

const char* host = « api.openweathermap.org »;

String api = « ***** »;

String city = « JANZE »;

//Pour l'utilisation du temps

#include <TimeLib.h>

//mise en place des ouvertures et fermetures par jour (1 = lundi)

// agenda[20][5] = numjour 0 à 6, ouvert(1)-fermé(0), H alarme 0 à 23, Min alarme 0 à 59, , H
fermeture/ouverture 0 à 23, Min fermeture/ouverture 0 à 59

// ex: 0,9,5,10,5,

int agenda[20][6] = {

    2,1,9,30,10,00,
    2,0,11,30,12,00,
    2,1,13,30,14,00,
    2,0,18,30,19,00,
    4,1,9,30,10,00,
    4,0,11,30,12,00,
    4,1,13,30,14,00,
    4,0,18,30,19,00,
    5,1,9,30,10,00,
    5,0,11,30,12,00,
    5,1,13,30,14,00,
    5,0,18,30,19,00,
    -1 //fin du fichier

};

```

```

unsigned long Epoch = 0; // timestamp de l'heure.

int Numjour = 0; //numero jour de la semaine avec 0 pour dimanche

String Heure = « 00:00 »; //hh:mm

int PartHeure = 0;

int PartMin = 0;

int LastPartHeure = 0;

String Date = « 00-00-0000 »;

String DateJM = « 00/00 »;

int NumAlarme = -1; //numéro d'alarme (utilisation pour [i])

int TypeAlarme = -1; //0 pour fermeture, 1 avant ouverture, 2 ouverture, 3 avant fermeture

String HeureAlarme = « 00:00 »;

String Message = » « ; // texte à afficher pendant l'alarme

String Jour = » « ;

String Horaires = « OUVERT LE MARDI, JEUDI ET VENDREDI DE 9h A 12h ET DE 14h A 19h »;

int meteo = 1;

float t = 0; //temp int

String ciel = » « ;

float temperatureC = 0;

//Mise en place de la matrice LED

#include <PxMatrix.h>

#include <Adafruit_GFX.h>

#include <fonts/FreeSans12pt7b.h>

#include <fonts/FreeSans9pt7b.h>

// Pins de la matrice LED

#ifdef ESP32

#define P_LAT 22

#define P_A 19

#define P_B 23

#define P_C 18

#define P_D 5

#define P_E 15

#define P_OE 2

hw_timer_t * timer = NULL;

portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;

#endif

#ifdef ESP8266

#include <Ticker.h>

```

```

Ticker display_ticker;

#define P_LAT 16

#define P_A 5

#define P_B 4

#define P_C 15

#define P_D 12

#define P_E 0

#define P_OE 2

#endif

#define matrix_width 64

#define matrix_height 32

uint8_t display_draw_time=20; //10-50 is usually fine

PxMATRIX display(64,32,P_LAT, P_OE,P_A,P_B,P_C,P_D);

// Les couleurs standards

uint16_t myRED = display.color565(255, 0, 0);
uint16_t myGREEN = display.color565(0, 255, 0);
uint16_t myBLUE = display.color565(0, 0, 255);
uint16_t myWHITE = display.color565(255, 255, 255);
uint16_t myYELLOW = display.color565(255, 255, 0);
uint16_t myCYAN = display.color565(0, 255, 255);
uint16_t myMAGENTA = display.color565(255, 0, 255);
uint16_t myBLACK = display.color565(0, 0, 0);

uint16_t myCOLORS[8]=
{myRED,myGREEN,myBLUE,myWHITE,myYELLOW,myCYAN,myMAGENTA,myBLACK};

#ifdef ESP8266

// ISR pour rafraichissement écran

void display_updater()

{
    display.display(display_draw_time);
}

#endif

#ifdef ESP32

void IRAM_ATTR display_updater(){

    // Increment the counter and set the time of ISR

    portENTER_CRITICAL_ISR(&timerMux);

    display.display(display_draw_time);

    portEXIT_CRITICAL_ISR(&timerMux);

}

```

```

#endif

void display_update_enable(bool is_enable)
{
#ifdef ESP8266
    if (is_enable)
        display_ticker.attach(0.002, display_updater);
    else
        display_ticker.detach();
#endif
#ifdef ESP32
    if (is_enable)
    {
        timer = timerBegin(0, 80, true);
        timerAttachInterrupt(timer, &display_updater, true);
        timerAlarmWrite(timer, 2000, true);
        timerAlarmEnable(timer);
    }
    else
    {
        timerDetachInterrupt(timer);
        timerAlarmDisable(timer);
    }
#endif
}

void setup() {
    Serial.begin(9600);
    //Connexion WIFI
    WiFi.begin(ssid, password);
    while ( WiFi.status() != WL_CONNECTED ) {
        delay ( 500 );
        Serial.print ( « . » );
    }
    Serial.println ( « CONNECTED » );
    //Affichage
    display.begin(16);
    display.setFastUpdate(true);
    display_update_enable(true);
}

```

```

// Demarrage client NTP
timeClient.begin();

//demarrage fht22
dht.begin();
}

void loop() {
if ((WiFi.status() == WL_CONNECTED)) { //Test connexion WIFI
    recupheure ();    //recuperation de l'heure
    //affichage sur port console
    Serial.print(« Jour : « );
    Serial.print(Numjour );
    Serial.print( » - Heure : « );
    Serial.println(Heure);
    Serial.print(« partie heure : « );
    Serial.print(PartHeure );
    Serial.print( » - partie min : « );
    Serial.println(PartMin);
    Serial.print(« date : « );
    Serial.print(Date);
    Serial.print( » - date pour recherches : « );
    Serial.println(DateJM);
    if (NumAlarme == -1){                // cherche alarme
        cherchealarme ();
    }
    if (NumAlarme != -1){                // cherche alarme si heure d'alarme atteint
        if (TypeAlarme == 0 || TypeAlarme == 2){
            if (PartHeure == agenda[NumAlarme][2] && PartMin == agenda[NumAlarme][3]){
                cherchealarme ();
            }
        }
        else {
            if (PartHeure == agenda[NumAlarme][4] && PartMin == agenda[NumAlarme][5]){
                cherchealarme ();
            }
        }
    }
    Serial.print(« num Alarme : « );

```

```

Serial.print(NumAlarme);
Serial.print( » - heure Alarme : « );
Serial.print(HeureAlarme );
Serial.print( » - type Alarme : « );
Serial.println(TypeAlarme );
affichage ();
Serial.print(« Texte bas : « );
Serial.println(Message);
Serial.println();
}
delay(200);
}

/*****
* récupération de l'heure *
*****/

void recupheure (){
    // Recup heure puis affichage
    int tempheure = PartHeure;
    timeClient.update();
    Numjour = timeClient.getDay(); // jour de la semaine
    time_t utcCalc = timeClient.getEpochTime() ;
    //declenchement récupération données météo dht22 et serveur
    if (PartHeure != hour(utcCalc)){
        recupmeteo ();
    }
    if (PartMin != minute(utcCalc)){
        recupdht22 ();
    }
    char buf[50];
    PartHeure = hour(utcCalc);
    PartMin =minute(utcCalc);
    snprintf(buf, sizeof(buf), »%02d:%02d », hour(utcCalc), minute(utcCalc));
    Heure = String(buf);
    snprintf(buf, sizeof(buf), »%02d-%02d-%4d », day(utcCalc), month(utcCalc), year(utcCalc));
    Date = String(buf);
    snprintf(buf, sizeof(buf), »%02d/%02d », day(utcCalc), month(utcCalc));
    DateJM = String(buf);

```

```

if (LastPartHeure != PartHeure){
    NumAlarme = -1;
}
LastPartHeure = PartHeure;
}

/*****

* Recherche meteo dht 22 *

*****/

void recupdht22 (){
    // Lecture de la température en Celcius
    t = dht.readTemperature();

    // Stop le programme et renvoie un message d'erreur si le capteur ne renvoie aucune mesure
    if (isnan(t)) {
        Serial.println(« Echec de lecture ! »);
        return;
    }
    Serial.print(« Temperature: « );
    Serial.print(t);
    Serial.print( » °C « );
}

/*****

* Recherche meteo serveur *

*****/

void recupmeteo (){
    WiFiClient client;
    delay (100);
    if (client.connect(host, 80))
    {
        Serial.println(« Envois de la requête meteo »);

        //Préparation et envois de la reqête vers l'API avec vos paramètre initialisé ci-dessus
        client.print(String(« GET /data/2.5/weather?q= » + city + « &APPID= » + api +
        « &lang=fr&units=metric » + » HTTP/1.1\r\n » +

            « Host: » + host + « \r\n » +

            « Connection: close\r\n » +

            « \r\n »

        ));

        /*
        *

```



```

* Récupération de la réponse
*/

char status[32] = {0}; //définition d'un char pour contenir la réponse à la requête
client.readBytesUntil('\r', status, sizeof(status));
if (strcmp(status, « HTTP/1.1 200 OK ») != 0) {
    Serial.print(F(« Réponse inattendue: « ));
    Serial.println(status);
    return;
}
else{
    Serial.println(« La réponse de la requête vient d'arriver »);
}

char endOfHeaders[] = « \r\n\r\n »;
if (!client.find(endOfHeaders)) {
    Serial.println(F(« Réponse non valide »));
    return;
}
else{
    Serial.println(« Traitement de la requête... »);
}
/*
* Mise en forme de la requête (afin qu'elle soit traitable dans l'IDE arduino)
*
*/

//la taille du buffer à créer -> pour le savoir tester sur le site : https://arduinojson.org/v5/assistant/
const size_t capacity =JSON_ARRAY_SIZE(1) + JSON_OBJECT_SIZE(1) + 2*JSON_OBJECT_SIZE(2) +
JSON_OBJECT_SIZE(4) + JSON_OBJECT_SIZE(5) + JSON_OBJECT_SIZE(6) + JSON_OBJECT_SIZE(12) + 390;

DynamicJsonBuffer jsonBuffer(capacity); //création du buffer

JsonObject& root = jsonBuffer.parseObject(client); //création du tableau qui va contenir la réponse à
notre requête

if (!root.success()) {
    Serial.println(F(« L'Analyse de la requête json a échouée! »));
    return;
}
else{
    Serial.println(« La requête a été correctement traitée, on passe au traitement des résultats »);
}
/*
* Partie non obligatoire, c'est juste pour faire stylée

```

```

*/

for (int i = 0; i<5; i++){
    Serial.print(« . »);
}
}

/*
* Traitement de la requête json
*
*/

//Parcours du tableau JSON reçus pour avoir les informations que l'on souhaite
const char* temps = root[« weather »][0][« description »];
ciel = String(temps);
ciel.toUpperCase();
temperatureC = root[« main »][« temp »];
Serial.println(« »);
//Affiche des informations sur le port
Serial.println(« La température à » + city + » est de » + temperatureC + » °C »);
Serial.println(« la météo à » + city + » est de type » + ciel);
client.stop();
}
}

/*****
* Recherche prochaine alarme *
*****/

void cherchealarme (){
    NumAlarme = -1;
    Serial.println(« recherche alarme »);
    for (int i=0 ; agenda[i][0] != -1 ; i++){
        if (agenda[i][0] == Numjour){ //recherche d'alarme pour le jour en cours
            if ( PartHeure < agenda[i][2] && NumAlarme == -1){
                NumAlarme = i;
                if (agenda[i][1] == 0){
                    TypeAlarme = 2;
                }
            }
            else {
                TypeAlarme = 0;
            }
        }
    }
}

```

```

}
else if ( PartHeure == agenda[i][2] && PartMin < agenda[i][3] && NumAlarme == -1){
    NumAlarme = i;
    if (agenda[i][1] == 0){
        TypeAlarme = 2;
    }
    else {
        TypeAlarme = 0;
    }
}
else if ( PartHeure < agenda[i][4] && NumAlarme == -1){
    NumAlarme = i;
    if (agenda[i][1] == 0){
        TypeAlarme = 3;
    }
    else {
        TypeAlarme = 1;
    }
}
else if ( PartHeure == agenda[i][4] && PartMin < agenda[i][5] && NumAlarme == -1){
    NumAlarme = i;
    if (agenda[i][1] == 0){
        TypeAlarme = 3;
    }
    else {
        TypeAlarme = 1;
    }
}
else if (Numjour < agenda[i][0] && NumAlarme == -1){ //si on ne trouve pas d'alarme, on prend la
première des jours suivants
    NumAlarme = i;
    if (agenda[i][1] == 0){
        TypeAlarme = 2;
    }
    else {
        TypeAlarme = 0;
    }
}

```

```

    }
}

if (NumAlarme == -1){ //si on ne trouve pas d'alarme, on prend la première
    NumAlarme = 0;
    if (agenda[0][1] == 0){
        TypeAlarme = 2;
    }
    else {
        TypeAlarme = 0;
    }
}

char buf[50];
sprintf(buf, sizeof(buf), »%02d:%02d », agenda[NumAlarme][4], agenda[NumAlarme][5]);
HeureAlarme = String(buf);
}

/*****
* Affichage sur la matrice *
*****/

void affichage (){
    if (NumAlarme != -1){
        switch (TypeAlarme){
            case 0:{
                if (agenda[NumAlarme][0] == 0)
                    Jour = « DIMANCHE »;
                else if (agenda[NumAlarme][0] == 1)
                    Jour = « LUNDI »;
                else if (agenda[NumAlarme][0] == 2)
                    Jour = « MARDI »;
                else if (agenda[NumAlarme][0] == 3)
                    Jour = « MERCREDI »;
                else if (agenda[NumAlarme][0] == 4)
                    Jour = « JEUDI »;
                else if (agenda[NumAlarme][0] == 5)
                    Jour = « VENDREDI »;
                else Jour = « SAMEDI »;
                Message = « OUVERTURE » + Jour + » A » + HeureAlarme ;
                scroll_text_double(17, 24, Heure, Message, myRED, myRED);
            }
        }
    }
}

```

```

break;}
case 1:{
    Message = « OUVERTURE A » + HeureAlarme;
    scroll_text_double(17, 24, Heure, Message , myYELLOW, myGREEN);
break;}
case 2:{
    scroll_text_double(17, 24, Heure, Horaires, myGREEN, myWHITE );
break;}
case 3:{
    int j = NumAlarme + 1;
    if (agenda[j][0] == -1){
        j=0;
    }
    if (agenda[j][0] == 0)
        Jour = « DIMANCHE »;
    else if (agenda[j][0] == 1)
        Jour = « LUNDI »;
    else if (agenda[j][0] == 2)
        Jour = « MARDI »;
    else if (agenda[j][0] == 3)
        Jour = « MERCREDI »;
    else if (agenda[j][0] == 4)
        Jour = « JEUDI »;
    else if (agenda[j][0] == 5)
        Jour = « VENDREDI »;
    else Jour = « SAMEDI »;
    char buf[50];
    snprintf(buf, sizeof(buf), »%02d:%02d », agenda[j][4], agenda[j][5]);
    String HeurAlarme2 = String(buf);
    Message = « Fermeture a » + HeureAlarme + » - prochaine ouverture » + Jour + » a » +
HeurAlarme2;
    scroll_text_double(17, 24, Heure, Message, myYELLOW, myRED);
break;}
}
}
}

/*****
* Scroll de l'affichage double *

```

```
*****/
```

```
void scroll_text_double(uint8_t y1pos, uint8_t y2pos, String textH, String textB, uint16_t colorH, uint16_t colorB)
```

```
{
  if (meteo == 1){
    textB = « T.int: » + String (t) + » C    T.ext: » + temperatureC + » C    Ciel: » + ciel;
    meteo = 0;
  }
  else {
    meteo = 1;
  }
  uint16_t textB_length = textB.length();
  display.setTextWrap(false); // we don't wrap text so it scrolls nicely
  display.setTextSize(1);
  display.setRotation(0);
  int x1pos = (matrix_width - (textH.length() * 11))/2 - 1;
  int xposdate = (matrix_width - (Date.length() * 6))/2;
  // affichage date en fixe avant scroll texte
  display.clearDisplay();
  display.setFont(& FreeSans12pt7b);
  display.setTextColor(colorH);
  display.setCursor(x1pos,y1pos);
  display.println (textH);
  display.setFont();
  display.setTextColor(colorB);
  if (TypeAlarme == 0){
    xposdate = (matrix_width - (5 * 6))/2;
    display.setCursor(xposdate, y2pos);
    display.println(« FERME »);
    delay(10000);
  }
  else {
    display.setCursor(xposdate, y2pos);
    display.println(Date);
    delay(5000);
  }
  // calcul du Scroll avec 5 pixels par caractères
  for (int x2pos = matrix_width; x2pos > -(textB_length * 6); x2pos--)
```

```
{  
  display.clearDisplay();  
  display.setFont(& FreeSans12pt7b);  
  display.setTextColor(colorH);  
  display.setCursor(x1pos,y1pos);  
  display.println (textH);  
  display.setFont();  
  display.setTextColor(colorB);  
  display.setCursor(x2pos, y2pos);  
  display.println(textB);  
  delay(30);  
  yield();  
  delay(30 / 5);  
  yield();  
}  
}
```

Résultat final