TUTORIEL : CALENDRIER BATEAU

1) Matériel nécessaire :

Ce projet consiste à faire un calendrier perpétuel ayant la forme d'un bateau. Les parties conceptions et montages sont disponibles dans la partie SolidWorks.

Pour ce projet on aura besoin :

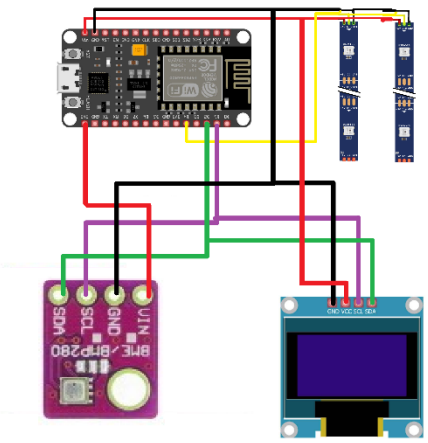
- D'une carte Arduino (ici esp8266).
- Un écran OLED I2C
- Un BME280 : capteur d'humidité, température et pression.
- Des rubans de LED (Ici des WS2812B)

La partie Arduino est ici principalement utilisé à des fins de décorations (ruban de LED) et d'ajout d'une horloge donnant aussi la température, pression et humidité de la pièce.

La partie calendrier en elle-même est purement mécanique.

2) Montage à faire :

On va faire le montage suivant :



3) Explication du code :

Comme tout code Arduino on va expliquer dans un premier temps les bibliothèques que l'on va utiliser :

- Wire.h : elle n'est pas indispensable en théorie cependant elle aide grandement à la stabilité des connections en I2C que l'on va utiliser ici.

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

- Adafruit_GFX.h : c'est une bibliothèque d'Arduino qui permet l'affichage graphique de formes basique et de différentes polices de caractère sur des écrans.

```
#include <Adafruit_GFX.h>
```

- Adafruit_SSD1306.h : est la bibliothèque spécifique au type d'écran OLED que l'on utilise

```
#include <Adafruit_SSD1306.h>
```

- Adafruit_BME280.h : est la bibliothèque spécifique au capteur BME280 (qui permet de capter l'humidité, la pression et la température de l'air)

```
#include <Adafruit_BME280.h>
```

- ESP8266WiFi.h : est la bibliothèque permettant la connexion au Wifi via une carte esp8266

```
#include <ESP8266WiFi.h>
```

- ESP8266WebServer.h : bibliothèques permettant la création d'un serveur via une esp8266

```
#include <ESP8266WebServer.h>
```

- WiFiManager.h : Bibliothèques qui permettent de plus facilement contrôler la connexion Wi-Fi de votre carte en permettant à l'utilisateur d'indiqué lequel utilisé parmi ceux disponible (au lieu devoir hard coder un nouveau à chaque changement de lieu)

```
#include <WiFiManager.h>
```

- time.h : Il permet d'avoir l'heure et la minute à laquelle nous sommes à un moment donnés grâce à des fonctions spécifiques.

```
#include <time.h>
```

- Adafruit_NeoPixel.h : Elle permettra de contrôler facilement les rubans de LED grâce à des fonctions prédéfinies.

```
#include <Adafruit_NeoPixel.h>
```

On doit ensuite initialiser les composant de notre montage :

- Pour l'écran : On va donner sa largeur (width en anglais) et sa hauteur (height) en anglais Ici on a un écran de 128 pixels par 64 pixels.

```
#define WIDTH 128
#define HEIGHT 64
```

Et ensuite on va déclarer l'écran dans sa propre ligne (ici avec le nom display)

```
Adafruit_SSD1306 display(WIDTH, HEIGHT, &Wire, -1);
```

Les valeurs à données sont largeur, hauteur, &Wire qui dit que la connexion sera faite en I2C et le -1 définissant la connexion pour remise à 0 (en l'occurrence ici le définissant comme n'existant pas car il n'y a pas de connexion avec numéros négatif)

- Pour les rubans de LEDs on va d'abord définir la prise sur laquelle on va le connecter :

```
#define LED_PIN D4
```

A noter qu'on peut définir 2 prises différentes si on souhaite des couleurs différentes sur chaque ruban.

Puis le type de LED utilisée :

```
#define LED_TYPE WS2812
```

Ainsi que le nombre de LED :

```
#define NUM_LED 4
```

Et l'ordre des couleurs RGB, car parfois certain ruban inverse les couleurs si on leurs donne les informations dans l'ordre, ici mon ruban inversait le rouge et le vert d'où l'ordre GRB.

```
#define COLOR_ORDER GRB
```

Enfin on initialise les rubans de LED sous le nom LED en donnant son nom et le nombre de LED du ruban, l'ordre des couleurs avec NEO_GRB et la fréquence de rafraichissement des LED avec NEOKHZ800 (ici 800kHz:

```
Adafruit_NeoPixel LED(NUM_LED,LED_PIN,NEO_GRB+NEO_KHZ800);
```

Si on souhaite contrôler les 2 rubans indépendamment il faudrait initialiser un autre ruban de LED, si le nombre de LED diffère entre les 2 rubans il est possible de donner des nombres différents.

- Pour le BME on va simplement déclarer dans le code qu'il existe :

```
Adafruit_BME280 bmp;
```

On va ensuite passer à l'initialisation utile pour la suite du code

- Tout d'abord le WiFiManager qui permet de choisir le réseau Wi-Fi sur lequel on se connecte parmi une liste de connexion internet reconnue par la carte :

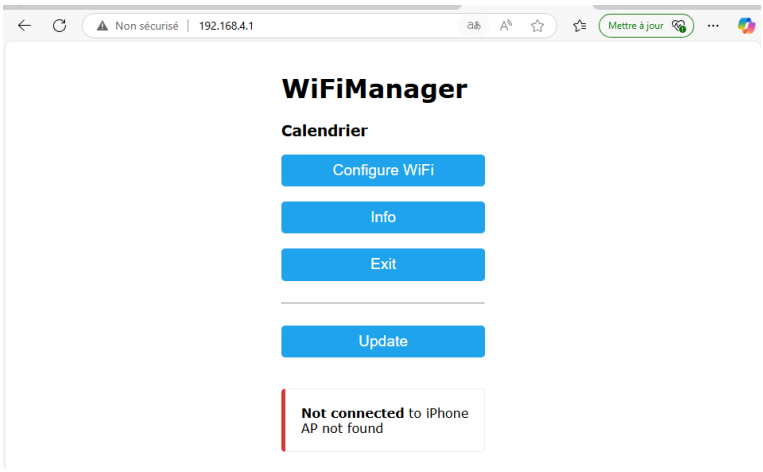
On commence par initialiser le client .

```
WiFiManager WM;
```

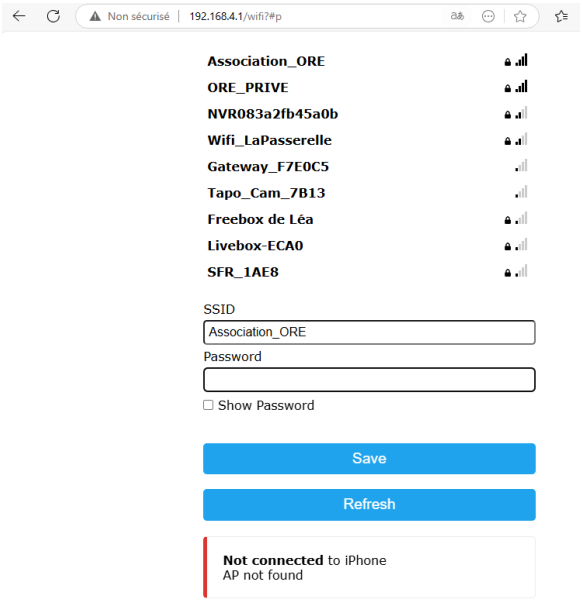
On définit ensuite le nom et mot de passe qu'on souhaite pour pouvoir se connecter à la carte.

```
const char* ssid = "Calendrier";
const char* password = "calendrier";
```

Une fois connecter à ce Wi-Fi sur votre portable ou PC, il devrait vous rediriger vers un site par défaut, sinon connectez-vous à : 192.168.4.1



Allez dans Configure Wi-Fi et choisissez le Wi-Fi sur lequel vous souhaitez connecter votre carte.



Après la 1ère utilisation le code utilisera par défaut la dernière connexion wifi en mémoire (sauf à s'il est remis à zéro de force à chaque lancement) et ne passera pas en mode émission de Wi-Fi sur lequel vous devrez vous connecter si le Wi-Fi en mémoire fonctionne.

La fonction permettant de reset est :

.resetSettings()

On définit ensuite les constantes utiles pour la suite du code :

- Le nombre d'heure de décalage avec le standard UTC.

```
const int decalage = 1;
```

- Le délai en secondes à mettre entre chaque demande au serveur pour l'heure (afin d'éviter que ce dernier refuse nos demandes car trop nombreuses) .

```
const int delaiDemande = 5 * 60;
```

- Ensuite on initialise 2 valeurs dans lesquelles seront stocké des temps en millisecondes à savoir le temps depuis la dernière demande au serveur et depuis le dernier changement de mode pour l'écran.

```
unsigned long derniereDemande = millis(); // moment de la plus récente demande au serveur NTP
unsigned long dernieremaJ = millis(); // moment de la plus récente mise à jour de l'affichage de l'heure
```

- Ainsi que 2 autres, heure courante qui permettra de faire tourner l'horloge en attendant la prochaine demande serveur et dernier_affichage qui permet de stocker le moment du dernier changement d'affiche entre l'heure et date en gros, et les valeurs du BME280

```
unsigned long heure_courante = millis();
unsigned long dernier_affichage = millis();
```

- Ensuite on définit une variable time_t qui est le nombre de seconde écoulées entre maintenant et le 1er janvier 1970 à 0 :00 UTC

```
time_t maintenant;
```

- Struct tm* est la structure la plus utilisé en C pour stocker des date sus la forme la plus compréhensible pour un humain avec jour, mois, année, heure, minutes et secondes. Elle servira dans la suite à stocké ce qui viens du serveur.

```
struct tm * timeinfo;
```

- On définit aussi un booléen qui servira à changer de type d'affichage entre l'heure et les valeurs du capteur BME280.

```
bool gros=false;
```

On va alors commencer la partie void setup() du code :

```
void setup() {
```

- On commence par lancer le moniteur série, cette étape n'est pas forcément nécessaire au bon fonctionnement du code, cependant cela facilite toute étape de débug par la suite en permettant l'affichage de ce que l'on souhaite coté IDE.

```
  Serial.begin(115200);
```

- On va ensuite lancer nos rubans de LED avec la ligne suivante, qui donne toutes les informations précédemment définis à la bibliothèques FastLED pour savoir utiliser notre ruban. Il est possible de mettre 2 lignes distingue en changent le nom (ici LED) et la connexion (LED_PIN) pour contrôler les 2 rubans indépendamment.

```
  FastLED.addLeds<LED_TYPE,LED_PIN,COLOR_ORDER>(LED,NUM_LED);
```

- On va ensuite lance l'écran qu'on avait défini plus haut en lui donnant le type et son adresse I2C (0x3C).

```
display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
```

L’adresse I2C est nécessaire car tous les objets connectés à une carte en I2C le sont par les même prises (SCL/SDA, fil vert et violet du schéma) tous peuvent donc recevoir les messages envoyés par ces prises, une adresse est alors nécessaire pour chaque appareil. Afin que la carte sache qui lui envoi un message et à un objet de savoir que ce message lui est destiné.

- On enlève tout message précédemment afficher sur l’écran.

```
 display.clearDisplay();
```

- On remet notre propre message sur l’écran et pour cela les fonctions sont à connaitre sont les suivantes (en plus du clear vu plus haut pour remettre l’écran à 0:

- .setTextSize : Donne la taille du texte (de 1 à 3)

```
display.setTextSize(3);
```

- .setTextColor : Pour choisir la couleur du texte (si l’écran permet plusieurs couleurs)

```
display.setTextColor(WHITE);
```

- .setCursor : Pour choisir la position (2 nombres) du pixel ou tu commenceras à écrire.

```
display.setCursor(20, 10);
```

- .print : Pour écrire un texte ou une variable

```
display.print("O R E ");
```

- .println : Pour écrire un texte ou une variable et revenir à la ligne avec un écart pertinent pour ta taille de caractère.

```
display.println("   En attente d'une ");
```

- .display : Pour afficher toutes les informations données depuis le dernier display.

```
display.display();
```

Dans notre cas ce morceau du code a pour but de faire comprendre à l’utilisateur que le bateau recherche une connexion wifi le temps que WiFiManager se connecte à celle en mémoire ou bien lance son propre Wi-Fi

- On passe ensuite sur la partie WiFiManager

On passe d’abord l’antenne Wi-Fi en mode émission

```
WiFi.mode(WIFI_STA);
```

On reste ensuite dans une boucle le temps « tant que » qui va attendre soit que le WiFiManager se connecte au Wi-Fi en mémoire soit que l'utilisateur donne une connexion valide à la carte. En attendant on affichera des points dans le serial moniteur pour débug et voir si le problème est à ce niveau.

```
while(!WM.autoConnect(ssid, password))
    {
     Serial.print(".");
    }
   Serial.print("Connecte");
```

Ensuite on sort l'information de temps du serveur ntp (Network Time Protocole)de l'université de Nice. La 1ère valeur donne le décalage en secondes par rapport à UTC, le suivant correspond au décalage heure d'été heure d'hiver ici on laisse à heure d'hiver par défaut et enfin on donne l'adresse du serveur auquel on se connecte.

```
configTime(decalage * 3600, 0, "ntp.unice.fr");
```

Cette boucle « tant que » permet juste de s'assurer que la synchronisation avec le serveur NTP a bien eu lieu en testant que le temps est bien arrivé au présent et pas bloqué à la date « 0 » qui est le 1er janvier 1970.

```
    while (time(nullptr) <= 100000) {
     // Serial.print(".");
     delay(200);
    }
```

Maintenant on stocke heure actuelle dans la variable maintenant défini vide précédemment :

```
   time(&maintenant);
```

On a ensuite une boucle de débug qui cherche à démarrer le BME280 et renvoyer un message d'erreur si le BME ne répond pas :

```
     while(!bmp.begin(0x76))
     {
       Serial.println("PAS TROUVE");
       delay(1000);
     }
```

On démarre enfin les LED et ont leurs donne leurs couleurs :

```
LED.begin();
ledScenario();
```

La fonction LED.begin est une fonction construite dans Adafruit Neo Pixel alors que ledScenario est défini dans le code explicitement à la fin de ce dernier. En commençant par void puis le nom de la fonction, d'une façon similaire à void setup et void loop. :

```
void ledScenario(){
  LED.setPixelColor(0,LED.Color(0,255,255));
  LED.setPixelColor(1,LED.Color(0,255,255));
  LED.setPixelColor(2,LED.Color(0,255,255));
```

```
  LED.setPixelColor(3,LED.Color(0,255,255));
  LED.show();
}
```

On y fixe la couleur de chaque led (dont l'index commence à 0) à la couleur bleu clair qui ressort si on maximise le bleu et le vert. Et on fait afficher toute nos modifications de couleurs avec la fonction .show()

On passe alors au void loop.

```
   void loop() {
```

On comme par définir la variable « heure courante » avec millis afin de calculer tous les délais par la suite par rapport au début de la boucle : (la fonction millis donne le nombre de milliseconde depuis le lancement de la carte)

```
heure_courante=millis();
```

On met tout d'abord une condition sur le temps depuis le dernier affichage de la date et l'heure actuelle en faisant la différence entre les 2 nombre millisecondes :

```
if((heure_courante-dernier_affichage)>2000)
  {
```

(ici on a une condition qui s'active toute les 2 secondes)

Alors on inverse le booléen « gros » qui change l'affichage :

```
gros=!gros;
```

On remet la variable dernier affichage à l'instant présent et on sort de la condition:

```
dernier_affichage=millis();
}
```

On commence alors une autre condition :

Si le booléen est alors « true » (vrai)

```
if (gros)
  {
```

alors on lance la fonction bigDisplay

```
   bigDisplay();
  }
```

Que l'on a défini, comme pour ledScenario, en fin de code comme suit :

```
void bigDisplay() {
```

Tout d'abord on remet les conditions de bases pour un bonne affichage sur l'écran : remettre à 0 ce qui pourrais déjà y être puis définir de quelle taille , couleur et où sur l'écran nous allons écrire.

```
display.clearDisplay();
display.setTextSize(3);
display.setTextColor(WHITE);
display.setCursor(20, 20);
```

Ensuite on définit la variable timeinfo avec les informations issues du serveur NTP, qui seront stocké sous une forme facilement lisible pour un humain.

```
timeinfo = localtime(&maintenant);
```

Ensuite on met une condition sur l'heure stockée dans le timeinfo. Si elle est inférieure à 10 ,alors on affiche un 0, ce qui permet de ne pas avoir de soucis de positionnement entre les heures avant et après 10 qui seront alors toujours écrite avec 2 groupes de 2 chiffres.

```
if ((timeinfo->tm_hour ) < 10) {
    display.print("0");
  }
```

Ensuite on affiche simplement l'heure (tm_hour) stocké dans la mémoire tel qu'elle est ainsi que «: » pour séparer les heures des minutes.

```
display.print(timeinfo->tm_hour );  // heure entre 0 et 23
display.print(":");
```

Puis on réitère exactement les mêmes étapes mais pour les minutes (tm_min) cette fois ci :

```
if (timeinfo->tm_min < 10) {
    display.print("0");
  }
  display.print(timeinfo->tm_min);
```

On finit enfin par envoyer notre heure mise en page sur l'écran avec la fonction prévue pour :

```
  display.display();
}
```

Cependant si le booléen « gros » n'est pas « true » alors on lance la fonction updateDisplay

```
else
  {
    updateDisplay();
  }
```

Comme les précédentes elle est définie en fin de code.
Cette dernière consiste en suite une suite de bloc tout identique dans leurs formes je ne vais donc en détailler qu'un seul. Elle sert à afficher les valeurs qui sortent du BME280 , à savoir température, pression et humidité de l'air, sur l'écran.

Comme à chaque fois que l'on cherche à écrire quelque chose sur l'écran on reprend les mêmes étapes de base :

```
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.setCursor(25,2);
```

On va ensuite écrire le nom de la variable qu'on veut écrire :

```
  display.print("Temperature : ");
```

Puis remettre le curseur ailleurs afin d'afficher la valeur elle-même . On change même de taille de caractère :

```
  display.setTextColor(WHITE);
  display.setCursor(20,32);
  display.setTextSize(2);
  display.print(bmp.readTemperature());
```

On utilise seulement « print » et pas « println » lorsqu'on écrit la valeur, afin mettre l'unité par la suite :

```
  display.println(" C");
```

On envoie ensuite les infos à l'écran. Puis on met un délai avant de commencer le bloc suivant pour une autre variable, qui correspond au temp d'affichage sur l'écran :

```
 display.display();
    delay(2000);
```

Et on recommence ce cycle pour chacune des données : température, pression , humidité et altitude.

On fait ensuite une condition sur l'écart entre l'heure actuelle grâce à millis et la dernière demande. Si le temps dépasse l'écart voulu entre les requêtes au serveur, alors on en fait une nouvelle et on redéfinit la dernière demande à l'instant actuel.

```
if ((millis() - derniereDemande) >=  delaiDemande * 1000 ) {
    time(&maintenant);
    derniereDemande = millis();
  }
```

Comme millis n'est pas infini et que l'horloge doit tourner à l'infini tant qu'alimenter on doit prévoir un contre à l'overflow d'où cette boucle. Qui fait en sorte de , si le millis est plus faible que le précédent, ce qui est impossible sans overflow, de remettre la dernière demande à l'instant T.

```
 if (millis() < derniereDemande ) {
    time(&maintenant);
```

```
    dernieredemande = millis();
  }
```

Et enfin une nouvelle boucle afin de mettre à jour l'heure pendant les 5 entre chaque demande.

```
 if ((millis() - dernieremaJ) >=   1000 ) {
    maintenant = maintenant + 5;

    dernieremaJ = millis();
  }
```

4) Code brut à recopier :

```
#include <Wire.h>

#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#include <Adafruit_BME280.h>


#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <WiFiManager.h>
#include <time.h>

#define WIDTH 128
#define HEIGHT 64

#include  <Adafruit_NeoPixel.h>
#define LED_PIN D4
#define NUM_LED 4
#define LED_TYPE WS2812B
#define COLOR_ORDER GRB


Adafruit_NeoPixel LED(NUM_LED,LED_PIN,NEO_GRB+NEO_KHZ800);

Adafruit_SSD1306 display(WIDTH, HEIGHT, &Wire, -1);

Adafruit_BME280 bmp;

WiFiManager WM;
const char* ssid = "Calendrier";
const char* password = "calendrier";

const int decalage = 1;
const int delaiDemande = 5 * 60;

unsigned long derniereDemande = millis(); // moment de la plus récente demande
au serveur NTP
unsigned long derniereMaJ = millis(); // moment de la plus récente mise à jour
de l'affichage de l'heure

time_t maintenant;
struct tm * timeinfo;

unsigned long heure_courante = millis();

bool gros=false;
```

```
void setup() {

  Serial.begin(115200);
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
  display.clearDisplay();
  display.setTextSize(3);
  display.setTextColor(WHITE);
  display.setCursor(20, 10);
  display.print("O R E ");
  display.setTextSize(1);
  display.setCursor(0,40);
  display.println("   En attente d'une ");
  display.setCursor(0,50);
  display.print("    connexion WiFi");
  display.display();
  delay(4000);

  WiFi.mode(WIFI_STA);
 while(!WM.autoConnect(ssid, password))
 {
  Serial.print(".");
 }
  Serial.print("Connecte");

 configTime(decalage * 3600, 0, "ntp.unice.fr");
 while (time(nullptr) <= 100000) {
  delay(1000);
 }
 time(&maintenant);
  while(!bmp.begin(0x76)){
    Serial.println("PAS TROUVE");
    delay(1000);
  }
LED.begin();
ledScenario();
}


void loop() {
  heure_courante=millis();
  if((heure_courante-dernier_affichage)>2000)
  {
    gros=!gros;
    dernier_affichage=millis();
  }
  if (gros)
  {
    bigDisplay();
    afficheHeureDate();
```

```
  }
  else
  {
    updateDisplay();
  }
if ((millis() - derniereDemande) >=  delaiDemande * 1000 ) {
    time(&maintenant);
    derniereDemande = millis();
  }
  // est-ce que millis() a débordé?
 if (millis() < derniereDemande ) {
    time(&maintenant);
    derniereDemande = millis();
  }
  // est-ce le moment de raffraichir la date indiquée?
  if ((millis() - dernieremaJ) >=   1000 ) {
    maintenant = maintenant + 5;

    dernieremaJ = millis();
  }
}


void bigDisplay() {
  display.clearDisplay();
  display.setTextSize(3);
  display.setTextColor(WHITE);
  display.setCursor(20, 20);
  timeinfo = localtime(&maintenant);
  if ((timeinfo->tm_hour ) < 10) {
    display.print("0");
  }
  display.print(timeinfo->tm_hour );  // heure entre 0 et 23
  display.print(":");
  if (timeinfo->tm_min < 10) {
    display.print("0");
  }
  display.print(timeinfo->tm_min);   // timeinfo->tm_min: minutes (0 - 59)
  display.display();
  delay(2000);
}

void updateDisplay() {
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.setCursor(25,2);
  display.print("Temperature : ");
  display.setTextColor(WHITE);
```

```
  display.setCursor(20,32);
  display.setTextSize(2);
  display.print(bmp.readTemperature());
  display.println(" C");
    display.display();
    delay(2000);

  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(30, 2);
  display.print("Pression:");
  display.setCursor(20,32);
  display.setTextSize(2);
  display.print(int(bmp.readPressure()/100.0));
  display.println(" hPa");
  display.display();
  delay(2000);

  display.clearDisplay();
   display.setTextSize(1);
  display.setCursor(30,2);
  display.print("Humidite :");
  display.setCursor(25,32);
  display.setTextSize(2);
  display.print(bmp.readHumidity());
  display.print(" %");
   display.display();
  delay(2000);
}

void afficheHeureDate() {
  display.clearDisplay();
  timeinfo = localtime(&maintenant);
  display.setCursor(30, 20);
  display.setTextSize(2);
  if (timeinfo->tm_mday < 10) {
    display.print("0");
  }
  display.print(timeinfo->tm_mday);  // timeinfo->tm_mday: jour du mois (1 -
31)
  display.print("-");
  if ((timeinfo->tm_mon + 1) < 10) {
    display.print("0");
  }
  display.print(timeinfo->tm_mon + 1);    // timeinfo->tm_mon: mois (0 - 11, 0
correspond à janvier)
  display.setCursor(35,35);
  display.print(timeinfo->tm_year + 1900);  // timeinfo->tm_year: tm_year
nombre d'années écoulées depuis 1900
```

```
  display.display();
  delay(2000);
}

void ledScenario(){
  LED.setPixelColor(0,LED.Color(0,255,255));
  LED.setPixelColor(1,LED.Color(0,255,255));
  LED.setPixelColor(2,LED.Color(0,255,255));
  LED.setPixelColor(3,LED.Color(0,255,255));
  LED.show();
}
```