```
// Replace with your network credentials
const char* ssid = "FABLAB 2.4";
const char* password = "MonPetitPonant";
const int nuageux =13;
const int pluie=12;
const int soleilNuage =14;
const int soleilPluie =27;
const int soleil =26;
const int Orageux =25;
const int neigeux =33;
const int brouillard = 32;
const int BP_Brest = 4;
const int BP_Paris = 16;
const int BP_Marseille =17;
const int BP_Lille =18;
bool mise_veille = false;
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>
#include <FastLED.h>

#define DATA_PIN 5
#define WIDTH 8
#define HEIGHT 8
#define NUM_LEDS (WIDTH * HEIGHT)
#define BRIGHTNESS 80

CRGB leds[NUM_LEDS];
String ancien_location ="";
String current_date;

String last_weather_update;
String temperature;
String humidity;
int is_day;
int timing_push;
int ancien_timer;
int timer = millis();
int weather_code = 0;
String weather_description;

// ---------- Mapping serpentin ----------
int XY(int x, int y) {
```

```
    x = (WIDTH - 1) - x;   // <-- miroir horizontal corrigé

    if (y % 2 == 0) return y * WIDTH + x;
    return y * WIDTH + (WIDTH - 1 - x);
}

void drawPixel(int x, int y, CRGB color) {
    if (x < 0 || x >= WIDTH || y < 0 || y >= HEIGHT) return;
    leds[XY(x, y)] = color;
}



//---------chiffre 3*5--------

// Police 3x5 : chaque chiffre fait 3 pixels de large sur 5 de haut
const uint16_t digits3x5[10] = {
  0b111101101101111, // 0
  0b010110010010111, // 1
  0b111001111100111, // 2
  0b111001111001111, // 3
  0b101101111001001, // 4
  0b111100111001111, // 5
  0b111100111101111, // 6
  0b111001010010010, // 7
  0b111101111101111, // 8
  0b111101111001111  // 9
};
// Fonction pour dessiner un chiffre 3x5
void drawDigit3x5(int digit, int offsetX, int offsetY, CRGB color) {
  if (digit < 0 || digit > 9) return;

  for (int y = 0; y < 5; y++) {
    for (int x = 0; x < 3; x++) {
      // On lit le bit correspondant (14 - position)
      int bitPos = 14 - (y * 3 + x);
      if (digits3x5[digit] & (1 << bitPos)) {
        drawPixel(x + offsetX, y + offsetY, color);
      }
    }
  }
}
```

```
void drawNumber2Digits(int value, CRGB color) {
  value = constrain(value, 0, 99);
  int dizaine = value / 10;
  int unite = value % 10;

  // Si le chiffre est < 10, on peut soit ne rien afficher à gauche,
soit un 0
  if (value >= 10) {
    drawDigit3x5(dizaine, 0, 1, color); // Positionné à x=0, y=1 pour
centrer verticalement
  }
  drawDigit3x5(unite, 4, 1, color);     // Positionné à x=4, y=1
}

// ---------- Affichage température ----------
void drawTemperature(int temp, String location) {
  FastLED.clear();
  CRGB couleurTemp = CRGB::White;
  if (location == "Brest") couleurTemp = CRGB::Blue;
  else if (location == "Paris") couleurTemp = CRGB::Green;
  else if (location== "Lille") couleurTemp = CRGB::Red;
  else couleurTemp = CRGB::Orange;
   drawNumber2Digits(temp,couleurTemp);
   FastLED.show();
}

// Replace with the latitude and longitude to where you want to get the
weather
String latitude = "48.390394";
String longitude = "-4.486076";
// Enter your location
String location = "Brest";
// Type the timezone you want to get the time for
String timezone = "Europe/Lisbon";

void IRAM_ATTR Paris(){
  if (location == "Paris"){}
  else {
  latitude = "48.511197";
  longitude = "2.205589";

  location = "Paris";
  mise_veille = false;
```

```
  timezone = "Europe/Lisbon";}
}

void IRAM_ATTR Brest(){
  if (location == "Brest"){}
  else{
  latitude = "48.390394";
  longitude = "-4.486076";

  location = "Brest";

  timezone = "Europe/Lisbon";}
  mise_veille = false;
}

void IRAM_ATTR Marseille(){
  if (location == "Marseille"){}
  else{
  latitude = "43.174200";
  longitude = "5.221920";
  mise_veille = false;
  location = "Marseille";

  timezone = "Europe/Lisbon";}
}

void IRAM_ATTR Lille(){
  if (location == "Lille"){}
  else{
  latitude = "50.371632";
  longitude = "3.024128";
  mise_veille = false;
  location = "Lille";

  timezone = "Europe/Lisbon";}
}
// Store date and time


// SET VARIABLE TO 0 FOR TEMPERATURE IN FAHRENHEIT DEGREES
#define TEMP_CELSIUS 1
```

```
#if TEMP_CELSIUS
  String temperature_unit = "";
  const char degree_symbol[] = "\u00B0C";
#else
  String temperature_unit = "&temperature_unit=fahrenheit";
  const char degree_symbol[] = "\u00B0F";
#endif


void allume_nuageux(int num){
  analogWrite(nuageux,num);
  analogWrite(pluie,0);
  analogWrite(soleilNuage,0);
  analogWrite(soleilPluie,0);
  analogWrite(soleil,0);
  analogWrite(Orageux,0);
  analogWrite(neigeux,0);
  analogWrite(brouillard,0);
}

void allume_pluie(int num){
  analogWrite(nuageux,0);
  analogWrite(pluie,num);
  analogWrite(soleilNuage,0);
  analogWrite(soleilPluie,0);
  analogWrite(soleil,0);
  analogWrite(Orageux,0);
  analogWrite(neigeux,0);
  analogWrite(brouillard,0);
}
void allume_soleilNuage(int num){
  analogWrite(nuageux,0);
  analogWrite(pluie,0);
  analogWrite(soleilNuage,num);
  analogWrite(soleilPluie,0);
  analogWrite(soleil,0);
  analogWrite(Orageux,0);
  analogWrite(neigeux,0);
  analogWrite(brouillard,0);
}
void allume_soleilPluie(int num){
  analogWrite(nuageux,0);
  analogWrite(pluie,0);
```

```
  analogWrite(soleilNuage,0);
  analogWrite(soleilPluie,num);
  analogWrite(soleil,0);
  analogWrite(Orageux,0);
  analogWrite(neigeux,0);
  analogWrite(brouillard,0);
}
void allume_soleil(int num){
  analogWrite(nuageux,0);
  analogWrite(pluie,0);
  analogWrite(soleilNuage,0);
  analogWrite(soleilPluie,0);
  analogWrite(soleil,num);
  analogWrite(Orageux,0);
  analogWrite(neigeux,0);
  analogWrite(brouillard,0);
}

void allume_neigeux(int num){
  analogWrite(nuageux,0);
  analogWrite(pluie,0);
  analogWrite(soleilNuage,0);
  analogWrite(soleilPluie,0);
  analogWrite(soleil,0);
  analogWrite(Orageux,0);
  analogWrite(neigeux,num);
  analogWrite(brouillard,0);
}
void allume_orageux(int num){
  analogWrite(nuageux,0);
  analogWrite(pluie,0);
  analogWrite(soleilNuage,0);
  analogWrite(soleilPluie,0);
  analogWrite(soleil,0);
  analogWrite(Orageux,num);
  analogWrite(neigeux,0);
  analogWrite(brouillard,0);
}
void allume_brouillard(int num){
  analogWrite(nuageux,0);
  analogWrite(pluie,0);
  analogWrite(soleilNuage,0);
  analogWrite(soleilPluie,0);
```

```
  analogWrite(soleil,0);
  analogWrite(Orageux,0);
  analogWrite(neigeux,0);
  analogWrite(brouillard,num);
}
/*
  WMO Weather interpretation codes (WW)- Code Description
  0 Clear sky
  1, 2, 3 Mainly clear, partly cloudy, and overcast
  45, 48  Fog and depositing rime fog
  51, 53, 55  Drizzle: Light, moderate, and dense intensity
  56, 57  Freezing Drizzle: Light and dense intensity
  61, 63, 65  Rain: Slight, moderate and heavy intensity
  66, 67  Freezing Rain: Light and heavy intensity
  71, 73, 75  Snow fall: Slight, moderate, and heavy intensity
  77  Snow grains
  80, 81, 82  Rain showers: Slight, moderate, and violent
  85, 86  Snow showers slight and heavy
  95 *  Thunderstorm: Slight or moderate
  96, 99 *  Thunderstorm with slight and heavy hail
*/
void get_weather_description(int code, int num) {
  switch (code) {
    case 0:
      if(is_day==1) {allume_soleil(num) ; }
      else { allume_soleil(num); }
      weather_description = "CLEAR SKY";
      break;
    case 1:
      if(is_day==1) { allume_soleilNuage(num); }
      else { allume_soleilNuage(num); }
      weather_description = "MAINLY CLEAR";
      break;
    case 2:
      allume_soleilNuage(num);
      weather_description = "PARTLY CLOUDY";
      break;
    case 3:
      allume_nuageux(num);
      weather_description = "OVERCAST";
      break;
    case 45:
      allume_brouillard(num);
```

```
    weather_description = "FOG";
    break;
  case 48:
    allume_brouillard(num);
    weather_description = "DEPOSITING RIME FOG";
    break;
  case 51:
    allume_soleilPluie(num);
    weather_description = "DRIZZLE LIGHT INTENSITY";
    break;
  case 53:
    allume_pluie(num);
    weather_description = "DRIZZLE MODERATE INTENSITY";
    break;
  case 55:
    allume_pluie(num);
    weather_description = "DRIZZLE DENSE INTENSITY";
    break;
  case 56:
    allume_soleilPluie(num);
    weather_description = "FREEZING DRIZZLE LIGHT";
    break;
  case 57:
    allume_pluie(num);
    weather_description = "FREEZING DRIZZLE DENSE";
    break;
  case 61:
    allume_pluie(num);
    weather_description = "RAIN SLIGHT INTENSITY";
    break;
  case 63:
    allume_pluie(num);
    weather_description = "RAIN MODERATE INTENSITY";
    break;
  case 65:
    allume_pluie(num);
    weather_description = "RAIN HEAVY INTENSITY";
    break;
  case 66:
    allume_pluie(num);
    weather_description = "FREEZING RAIN LIGHT INTENSITY";
    break;
  case 67:
```

```
    allume_pluie(num);
    weather_description = "FREEZING RAIN HEAVY INTENSITY";
    break;
  case 71:
    allume_pluie(num);
    weather_description = "SNOW FALL SLIGHT INTENSITY";
    break;
  case 73:
    allume_neigeux(num);
    weather_description = "SNOW FALL MODERATE INTENSITY";
    break;
  case 75:
    allume_neigeux(num);
    weather_description = "SNOW FALL HEAVY INTENSITY";
    break;
  case 77:
    allume_neigeux(num);
    weather_description = "SNOW GRAINS";
    break;
  case 80:
    allume_orageux(num);
    weather_description = "RAIN SHOWERS SLIGHT";
    break;
  case 81:
    allume_orageux(num);
    weather_description = "RAIN SHOWERS MODERATE";
    break;
  case 82:
    allume_orageux(num);
    weather_description = "RAIN SHOWERS VIOLENT";
    break;
  case 85:
    allume_pluie(num);
    weather_description = "SNOW SHOWERS SLIGHT";
    break;
  case 86:
    allume_pluie(num);
    weather_description = "SNOW SHOWERS HEAVY";
    break;
  case 95:
    allume_orageux(num);
    weather_description = "THUNDERSTORM";
    break;
```

```
    case 96:
      allume_orageux(num);
      weather_description = "THUNDERSTORM SLIGHT HAIL";
      break;
    case 99:
      allume_orageux(num);
      weather_description = "THUNDERSTORM HEAVY HAIL";
      break;
    default:
      weather_description = "UNKNOWN WEATHER CODE";
      break;
  }
}

void get_weather_data() {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;
    // Construct the API endpoint
    String url =
String("http://api.open-meteo.com/v1/forecast?latitude=" + latitude +
"&longitude=" + longitude +
"&current=temperature_2m,relative_humidity_2m,is_day,precipitation,rain
,weather_code" + temperature_unit + "&timezone=" + timezone +
"&forecast_days=1");
    http.begin(url);
    int httpCode = http.GET(); // Make the GET request

    if (httpCode > 0) {
      // Check for the response
      if (httpCode == HTTP_CODE_OK) {
        String payload = http.getString();
        Serial.println("Request information:");
        Serial.println(payload);
        // Parse the JSON to extract the time
        JsonDocument doc;
        DeserializationError error = deserializeJson(doc, payload);
        if (!error) {
          const char* datetime = doc["current"]["time"];
          temperature = String(doc["current"]["temperature_2m"]);
          humidity = String(doc["current"]["relative_humidity_2m"]);
          is_day = String(doc["current"]["is_day"]).toInt();
          weather_code =
String(doc["current"]["weather_code"]).toInt();
```

```
          /*Serial.println(temperature);
          Serial.println(humidity);
          Serial.println(is_day);
          Serial.println(weather_code);
          Serial.println(String(timezone));*/
          // Split the datetime into date and time
          String datetime_str = String(datetime);
          int splitIndex = datetime_str.indexOf('T');
          current_date = datetime_str.substring(0, splitIndex);
          last_weather_update = datetime_str.substring(splitIndex + 1,
splitIndex + 9); // Extract time portion
        } else {
          Serial.print("deserializeJson() failed: ");
          Serial.println(error.c_str());
        }
      }
    } else {
      Serial.printf("GET request failed, error: %s\n",
http.errorToString(httpCode).c_str());
    }
    http.end(); // Close connection
  } else {
    Serial.println("Not connected to Wi-Fi");
  }
}

void setup() {
  Serial.begin(115200);
  pinMode(nuageux, OUTPUT);
  pinMode(pluie, OUTPUT);
  pinMode(soleilNuage, OUTPUT);
  pinMode(soleilPluie, OUTPUT);
  pinMode(soleil, OUTPUT);
  pinMode(Orageux, OUTPUT);
  pinMode(neigeux, OUTPUT);
  pinMode(brouillard, OUTPUT);
  pinMode(BP_Brest,INPUT_PULLUP);
  pinMode(BP_Paris,INPUT_PULLUP);
  pinMode(BP_Marseille,INPUT_PULLUP);
  pinMode(BP_Lille,INPUT_PULLUP);

  // bouton des villes
  attachInterrupt(BP_Brest, Brest, CHANGE);
```

```
  attachInterrupt(BP_Paris, Paris, CHANGE);
  attachInterrupt(BP_Marseille, Marseille, CHANGE);
  attachInterrupt(BP_Lille, Lille, CHANGE);
  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  Serial.print("Connecting");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  FastLED.addLeds<WS2812B, DATA_PIN, GRB>(leds, NUM_LEDS);
  FastLED.setBrightness(BRIGHTNESS);
  Serial.print("\nConnected to Wi-Fi network with IP Address: ");
  Serial.println(WiFi.localIP());

  // Create a display object
}

void loop() {
  timer = millis();
  if (ancien_location != location ){
    get_weather_data();
    get_weather_description(weather_code, 50);
    int tempInt = temperature.toInt();
    Serial.print(latitude);
    Serial.print(longitude);
    Serial.print(location);
    Serial.println(timezone);
    Serial.print("Température reçue: ");
    Serial.println(tempInt);

    Serial.println(weather_description);
    drawTemperature(tempInt,location);
    timing_push = millis();
    ancien_location = location;
  }
  if (timer >= ancien_timer + 6000){
    get_weather_data();
    if (mise_veille == true){
      get_weather_description(weather_code, 0);
    }else{get_weather_description(weather_code,50);}
    int tempInt = temperature.toInt();
    Serial.print(latitude);
```

```
    Serial.print(longitude);
    Serial.print(location);
    Serial.println(timezone);
    Serial.print("Température reçue: ");
    Serial.println(tempInt);

    Serial.println(weather_description);
    drawTemperature(tempInt,location);
    ancien_timer = millis();
  }
  if (timer >= timing_push + 15000 ){
    mise_veille = true;
    get_weather_description(weather_code, 0);
  }

}
```