EJP

Petit projet pour afficheur EJP EDF connecté

Code

```
#include <Arduino.h>
#include <WiFi.h>
#include <NTPClient.h>
#include <WiFiUdp.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>

const char *ssid = "ssid";
const char *password = "password";

const long gmtOffset = 3600; // GMT offset in seconds (same as UTC + 1)

WiFiUDP ntpUDP;
// Setup NTP server
NTPClient timeClient(ntpUDP, "pool.ntp.org", gmtOffset);

// Base URL for EDF API Call
const char *baseUrl = "https://api-
commerce.edf.fr/commerce/activet/v1/calendrier-jours-effacement?
option=EJP&identifiantConsommateur=src";

// GPIO Setup
const int todayLedPin = 12;
const int tomorrowLedPin = 13;
const int errorLedPin = 4;

// Check at desired times, 24H format, default 16, 18, 20, modify as
needed
const int checkTimes[] = {16, 18, 20}; // 4, 6, 8pm
const int minuteOffset = 0; // Smaller precision for fetch timing eg. if
set to 30 then check will be at checkTimes + 30min
const int lengthCheckTimes = sizeof(checkTimes) / sizeof(checkTimes[0]);

const int rateLimit = 10; // Request rate limit
const int repeatDelay = 1000; // How much time we wait between each
request
bool errorOccured = true;

void animateError() {
  digitalWrite(errorLedPin, HIGH);
  delay(100);
  digitalWrite(errorLedPin, LOW);
  delay(5000);
}
```

```
void animateWifi() {
  digitalWrite(todayLedPin, LOW);
  digitalWrite(tomorrowLedPin, LOW);
  digitalWrite(errorLedPin, LOW);
  for (int i = 0; i < 5; ++i) {
    digitalWrite(todayLedPin, HIGH);
    delay(50);
    digitalWrite(tomorrowLedPin, HIGH);
    delay(50);
    digitalWrite(todayLedPin, LOW);
    delay(50);
    digitalWrite(tomorrowLedPin, LOW);
    delay(50);
  }
}

void animateSuccess() {
  digitalWrite(todayLedPin, LOW);
  digitalWrite(tomorrowLedPin, LOW);
  digitalWrite(errorLedPin, LOW);
  for (int i = 0; i < 5; ++i) {
    digitalWrite(todayLedPin, HIGH);
    delay(100);
    digitalWrite(todayLedPin, LOW);
    digitalWrite(tomorrowLedPin, HIGH);
    delay(100);
    digitalWrite(tomorrowLedPin, LOW);
    digitalWrite(errorLedPin, HIGH);
    delay(100);
    digitalWrite(errorLedPin, LOW);
  }
  for (int i = 0; i < 3; ++i) {
    digitalWrite(todayLedPin, HIGH);
    digitalWrite(tomorrowLedPin, HIGH);
    delay(50);
    digitalWrite(todayLedPin, LOW);
    digitalWrite(tomorrowLedPin, LOW);
    delay(50);
  }
}

void fetchAndProcessEjpData() {
  Serial.println("Checking for EJP days...");

  // Initiate WiFi connection
  WiFi.begin(ssid, password);

  for (int i = 0; i < rateLimit && WiFi.status() != WL_CONNECTED; ++i) {
    delay(repeatDelay);
    Serial.println("Connecting to WiFi...");
  }

  if (WiFi.status() == WL_CONNECTED) {
```

```
    Serial.println("WiFi connected!");
    Serial.print("IP Address: ");
    Serial.println(WiFi.localIP());
    errorOccured = false;
    animateWifi();
  } else {
    Serial.println("Failed to connect to WiFi");
    errorOccured = true;
    return;
  }

  // Init and sync time
  timeClient.begin();
  timeClient.update();

  // Get UNIX time
  time_t epochTime = timeClient.getEpochTime();

  // Convert to easily readable struct
  struct tm *timeinfo;
  timeinfo = localtime(&epochTime);

  // Get the current year (add 1900)
  int currentYear = 1900 + timeinfo->tm_year;

  // Formatting to URL params
  char inferiorLimit[11];
  snprintf(inferiorLimit, sizeof(inferiorLimit), "%04d-%01d-%01d",
currentYear - 1, timeinfo->tm_mon + 1, timeinfo->tm_mday + 1);
  char superiorLimit[11];
  snprintf(superiorLimit, sizeof(superiorLimit), "%04d-%01d-%01d",
currentYear, timeinfo->tm_mon + 1, timeinfo->tm_mday + 1);

  // Construct the full API url
  String apiUrl = String(baseUrl) + "&dateApplicationBorneInf=" +
inferiorLimit + "&dateApplicationBorneSup=" + superiorLimit;
  Serial.println(apiUrl);

  // Data fetching
  HTTPClient http;

  bool success = false;

  for (int retry = 0; retry < rateLimit && !success; ++retry) {
    if (http.begin(apiUrl)) {
      int httpCode = http.GET();
      if (httpCode == HTTP_CODE_OK) {
        String payload = http.getString(); // Formatting

        // Parsing payload
        JsonDocument doc;
        DeserializationError error = deserializeJson(doc, payload);

        if (!error) {
```

```
          bool isEjpToday = false;
          bool isEjpTomorrow = false;

          // Access calendar data
          const int calendarLength = doc["content"]["options"][0]
["calendrier"].size();
          const char *tomorrowStatus = doc["content"]["options"][0]
["calendrier"][calendarLength-1]["statut"];
          const char *todayStatus = doc["content"]["options"][0]
["calendrier"][calendarLength-2]["statut"];

          // EJP check
          if (strcmp(tomorrowStatus, "EJP") == 0) {
            isEjpTomorrow = true;
          }
          if (strcmp(todayStatus, "EJP") == 0) {
            isEjpToday = true;
          }

          // Printing values for now, connecting to LED later
          Serial.print("Is EJP Today: ");
          Serial.println(isEjpToday);
          Serial.print("Is EJP Tomorrow: ");
          Serial.println(isEjpTomorrow);

          digitalWrite(todayLedPin, isEjpToday ? HIGH : LOW);
          digitalWrite(tomorrowLedPin, isEjpTomorrow ? HIGH : LOW);

          success = true;
        } else {
          Serial.println("Error parsing JSON");
        }
      } else {
        Serial.println("Error in HTTP request");
      }
      http.end();
    } else {
      Serial.println("Failed to connect to API");
    }
    delay(repeatDelay);
  }

  errorOccured = !success;

  WiFi.disconnect(true);
}

void setup() {
  Serial.begin(115200);

  // Init led state
  pinMode(todayLedPin, OUTPUT);
  pinMode(tomorrowLedPin, OUTPUT);
  pinMode(errorLedPin, OUTPUT);
```

```
  // Initial data retrieval at setup, easier debugging
  fetchAndProcessEjpData();

  if (!errorOccured) {
    animateSuccess();
  }
}

void loop() {
  // loop content

  // Check for EJP days at specified times
  for (int i = 0; i < lengthCheckTimes; ++i) {
    if (timeClient.getHours() == checkTimes[i] && timeClient.getMinutes()
== minuteOffset) {
      fetchAndProcessEjpData();
      if (!errorOccured) {
        animateSuccess();
      }
      // Prevent calling two times during the same minute
      delay(60000);
    }
  }

  if (errorOccured) {
    animateError();
  }
}
```