Tuto – Station d'arrosage ESP32 (avec météo, BME280, sonde de sol, relais, MP3 et page Web)

Ce tutoriel explique **comment fonctionne le code**, **comment le câbler**, **comment l'installer**, et **comment l'utiliser/diagnostiquer**.

---

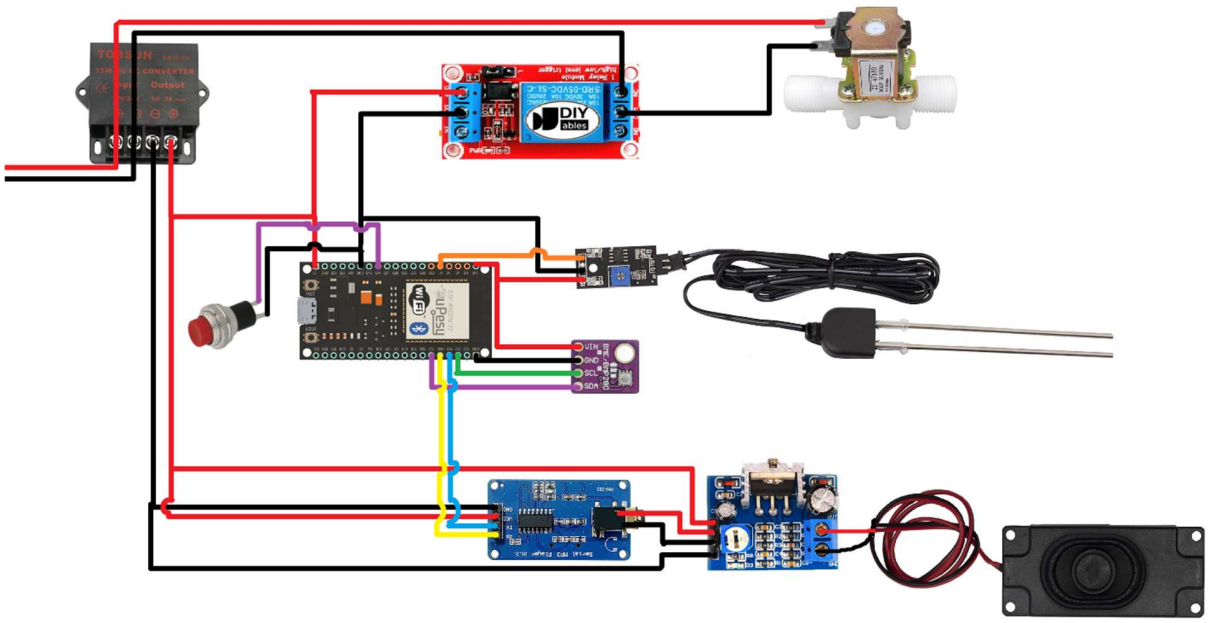
1) Matériel & câblage

Carte: ESP32 (DevKit v1 ou équivalent)

Capteurs/acteurs - **BME280** I²C 3,3 V (Adafruit/compatibles) — par défaut ESP32: `SDA = 21`, `SCL = 22`
- **Sonde humidité sol analogique** → pin **34** (entrée ADC uniquement)
> Note : privilégier une **sonde capacitive** 3,3 V (moins sujette à la corrosion et au bruit)
- **Relais électrovanne** → **PIN_RELAY = 27** *(par défaut dans le code)*
> ⚠️ Le commentaire d'en-tête évoquait pin 26 ; **le code utilise 27**. Modifiez `PIN_RELAY` si vous préférez 26.
> Alim de la bobine relais **isolée** du 3,3 V de l'ESP32, **masse commune**. Utiliser un module relais 5 V à optocoupleur est recommandé.
- **Bouton poussoir** → **pin 14** vers **GND** (utilise `INPUT_PULLUP`)
- **Lecteur MP3 série type YX5200/YX5300/DFPlayer**
`Module TX → ESP32 RX2 (GPIO16)`, `Module RX → ESP32 TX2 (GPIO17)`, GND commun, Vcc 5 V.
Pistes nommées `0001.mp3`, `0002.mp3`, … à la racine de la carte SD/USB.

Alimentation - ESP32 en 5 V.
- Relais et électrovanne doivent avoir une **alimentation dédiée** dimensionnée.

- **Masse commune** entre ESP32 et modules.



---

2) Bibliothèques Arduino à installer

- **Adafruit BME280 Library** (installe aussi Adafruit Unified Sensor / BusIO)
- **ArduinoJson** ≥ 6.x

IDE : Arduino IDE (ESP32 core) ou PlatformIO. Vitesse Moniteur Série : **115200 bauds**.

---

3) Paramètres à connaître / personnaliser

Dans le haut du code :

- **Wi-Fi**
  `const char* WIFI_SSID`, `WIFI_PASS` → vos identifiants Wi-Fi.
- **Localisation météo** via Open-Meteo/Météo-France :
  `OWM_LAT`, `OWM_LON` (latitude/longitude de votre jardin).
  Rafraîchissement: `OWM_REFRESH_MS` (par défaut 2 h).
- **Seuils d'arrosage (modifiables aussi via page Web)**
  - `SOIL_DRY_PCT_THRESH` (% d'humidité sol sous lequel on arrose)

- `RAIN_POP_THRESH` (probabilité de pluie [0..1] au-dessus de laquelle on *évite* d'arroser)
- `RAIN_MM_THRESH` (cumul pluie 12 h en mm au-dessus duquel on *évite* d'arroser)
- `WATERING_DURATION_S` (durée d'un cycle auto)

- **Calibration sonde de sol**
  - `SOIL_ADC_WET`: valeur ADC quand **très humide**
  - `SOIL_ADC_DRY`: valeur ADC quand **très sec**
- **Relais / bouton / ADC**
  - `PIN_RELAY` (par défaut **27**) et `RELAY_ACTIVE_LEVEL` (HIGH ou LOW selon votre module)
  - `PIN_BUTTON = 14` (vers GND, avec PULLUP)
  - `PIN_SOIL = 34`
- **Sécurités**
  - `MANUAL_LOCK_MS = 10 min` → empêche un redémarrage auto juste après un arrêt manuel
  - `MAX_WATER_MS = 30 min` → **coupe de sécurité** pour éviter un arrosage infini

---

4) Architecture du code (vue d'ensemble)

4.1 Services

- **Serveur Web** (port 80) → UI de supervision/commande + endpoints API
- **Client HTTP** → requête **Open-Meteo** (modèle Météo-France) pour POP et pluie horaire 12 h
- **UART2** → contrôle du module **MP3** via protocole YX5200/5300/DFPlayer

4.2 Capteurs

- **BME280** (T°, humidité, pression) en I^2C
  Adresses tentées : 0x76 puis 0x77
- **Sonde sol** : lecture **analogique** (0..4095) → transformée en **% d'humidité** via `soilPctFromRaw()`

4.3 Boucle de contrôle `controlLoop()`

1. Lit **sol** et **BME280**

2. Rafraîchit la **météo** toutes les `OWM_REFRESH_MS`

3. Applique la **logique d'arrosage** :
   - `soilDry = (soilPct < SOIL_DRY_PCT_THRESH)`
   - `rainSoon = (popMax12h ≥ RAIN_POP_THRESH) || (rainSum12h ≥ RAIN_MM_THRESH)`
   - Démarre si `soilDry` **et pas** `rainSoon`
   - Arrête à `WATERING_DURATION_S` ou si `MAX_WATER_MS` atteint (sécurité)

En **manuel**, la boucle **n'auto-déclenche pas**. On peut démarrer/arrêter via page Web.

4.4 Journalisation (Moniteur Série)

- `logStatus(src)` affiche périodiquement : T°, %HR, hPa, SoilRaw, Soil%, POPmax12h, Rain12h, Auto/Water, Bouton, RSSI.
- `logWaterEvent(msg)` trace les événements : *démarrage/arrêt/ sécurité*.
- Détection bouton : `[BTN] Press detected` / `Release`.

Exemple de lignes :

```
[STAT][23s][AUTO] T=24.3°C H=51.2% P=1013hPa | SoilRaw=2870 Soil=38% | POPmax
12h=20% Rain12h=0.0mm | Auto=ON Water=OFF | Button=released | RSSI=-62 dBm
[WATER] DÉMARRAGE arrosage (auto) | relay=ON | durAuto=300s | max=1800000ms
[AUDIO] Play 2
```

---

5) Interface Web & API

5.1 Page Web

- URL : `http://<IP de l'ESP32>/` (affichée au boot dans le Moniteur Série)
- Cartes : **Capteurs**, **Météo (12 h)**, **Arrosage**, **Audio**, **Réglages**
- Rafraîchissement auto toutes **3 s**

5.2 Endpoints API

- `GET /api/data` → JSON snapshot (capteurs, météo, réglages, état, RSSI)

- `POST /api/config`
  - body : `toggleMode=1` (bascule AUTO/MANUEL)
  - ou `th_soil`, `th_pop`, `th_mm`, `dur` pour enregistrer les seuils
- `POST /api/manual`
  - body : `water=on` ou `water=off`
  - `off` déclenche un **verrou manuel** de 10 min (évite redémarrage auto immédiat)
- `POST /api/audio`
  - body : `track=<n>` (ex. `1`, `2`, `3` ou `0` pour stop)
- `POST /api/refreshForecast` → force l'actualisation météo maintenant

Astuce test rapide (depuis un PC sur le même réseau) :

```
curl http://ESP32_IP/api/data
curl -d 'toggleMode=1' http://ESP32_IP/api/config
curl -d 'water=on' http://ESP32_IP/api/manual
curl -d 'track=1' http://ESP32_IP/api/audio
```

---

6) Météo : Open-Meteo (Météo-France)

- Endpoint utilisé : `meteofrance` avec `hourly=precipitation_probability,precipitation`
- Paramètres : `latitude`, `longitude`, `forecast_hours=12`, `timezone=auto`
- Traitement :
  - **POP max 12 h** (convertie de % → 0..1)
  - **Pluie cumulée 12 h** (mm)
  - Conditions « pluie bientôt » si `POPmax ≥ RAIN_POP_THRESH` **ou** `rainSum ≥ RAIN_MM_THRESH`
- Logs utiles : `[WEATHER] HTTP ...`, `[WEATHER] JSON error: ...`, puis récapitulatif `POPmax12h`, `Rain12h`.

---

7) Calibrer la sonde d'humidité du sol

1. **À l'air libre** (sonde sèche) : notez `ADC_DRY` ≈ lecture sur le Moniteur Série (`SoilRaw`).

2. **Plongée dans l'eau** (ou terre très humide) : notez `ADC_WET`.

3. Mettez ces valeurs dans :

   ```
   int SOIL_ADC_WET = <mesure humide>;
   int SOIL_ADC_DRY = <mesure sèche>;
   ```

4. Vérifiez que la **convertisseur en %** (`soilPctFromRaw`) donne un pourcentage plausible.

Si la sonde **capacititve** est alimentée en 3,3 V, l'ADC sera moins bruité. Évitez les sondes **résistives** à long terme (corrosion).

---

8) Relais & sécurité

- `RELAY_ACTIVE_LEVEL` selon votre module (certains sont actifs à l'état **LOW**).

- Limites : `WATERING_DURATION_S` (durée par cycle) et `MAX_WATER_MS` (coupure de sécurité).

- Après `manual off`, un **verrou** (`MANUAL_LOCK_MS`) empêche le démarrage auto pendant 10 min.

Toujours tester le relais **sans l'électrovanne** au début. Vérifiez que ON/OFF agit dans le bon sens.

---

9) Bouton & Audio

- Bouton **pin 14 -> GND** (pull-up interne).

- Détection avec **antirebond** (50 ms).

- À chaque **appui** : incrémente `currentTrack` et lance la piste suivante (`0001.mp3` → `0020.mp3`).

- API audio : `/api/audio` (`track=n`) ou bouton sur l'UI.

---

10) Installation (upload)

1. Ouvrez le sketch dans **Arduino IDE** (ESP32 core installé) ou **PlatformIO**.
2. Renseignez **SSID/Mot de passe Wi-Fi** dans le code.
3. Sélectionnez la **carte** ESP32 et le **port série**.
4. **Téléversez**.
5. Ouvrez le **Moniteur Série @115200** : notez l'IP affichée, les logs capteurs/météo, etc.
6. Naviguez sur `http://<IP>` pour l'UI Web.

---

11) Dépannage rapide

- **BME280 non détecté** → vérifiez câblage, alim 3,3 V, adresse 0x76/0x77, lignes SDA/SCL (21/22).
- **Météo KO** (`HTTP ...` ou `JSON error`) → connexion Wi-Fi, accès Internet, coordonnées valides.
- **Sonde % incohérent** → recalibrez `SOIL_ADC_WET/DRY`, vérifiez GND commun, câble, bruit.
- **Relais inversé** → ajustez `RELAY_ACTIVE_LEVEL` (HIGH ↔ LOW).
- **Page Web inaccessible** → IP, pare-feu, port 80, réseau identique PC/ESP32.
- **MP3 muet** → volume, nommage fichiers (`0001.mp3`), carte SD, câblage UART2 (16/17), GND commun.

---

12) Sécurité & bonnes pratiques

- **Ne publiez pas** vos identifiants Wi-Fi en clair sur Internet.
- Évitez d'exposer directement l'ESP32 sur Internet (HTTP non chiffré).

- Protégez l’alimentation et le relais (diode de roue libre si bobine, module relais de qualité, boîtier étanche).
- Préférez des **capteurs étanches** et **connecteurs** adaptés en extérieur.

---

13) Améliorations possibles

- **Sauvegarde des réglages** dans la NVS/`Preferences` (persistants après reboot).
- **Fenêtres horaires** d’arrosage (ex. uniquement la nuit).
- **mDNS** (`http://arrosage.local/`) et **OTA** (mise à jour Wi-Fi).
- **Filtre médian**/moyenne glissante sur la sonde sol pour plus de stabilité.
- **Alarme fuites** : compteur d’eau/flow-meter pour couper si débit anormal.
- **Économie énergie** : deep-sleep + réveil périodique (si pas besoin du serveur en continu).
- **Authentification** basique sur les endpoints si réseau partagé.

---

14) Références rapides (extraits)

Pins (par défaut)

```
SDA=21, SCL=22 | Soil ADC=34 | Relay=27 | Button=14 | MP3: RX2=16, TX2=17
```

Seuils

```
SOIL_DRY_PCT_THRESH=35%, RAIN_POP_THRESH=0.5, RAIN_MM_THRESH=1.0mm, WATERING_DURATION_S=300s
```

Sécurités

```
MANUAL_LOCK_MS=10min, MAX_WATER_MS=30min
```

Endpoints

```
/           (UI Web)
/api/data   (GET)
/api/config (POST: toggleMode | th_soil | th_pop | th_mm | dur)
/api/manual (POST: water=on/off)
/api/audio  (POST: track=n, 0=stop)
/api/refreshForecast (POST)
```

```
/*
  Projet: Station d'arrosage ESP32 + BME280 + sonde humidité sol + relais électrovanne
        + prévisions de pluie (Open-Meteo / Météo-France) + lecteur MP3 série (type YX5300/DFPlayer)
        + page Web de supervision/commande.

  Auteur: ChatGPT (GPT-5 Thinking)
  Cible:  ESP32 (Arduino framework)

  Matériel attendu:
   - ESP32 (DevKit v1 ou équivalent)
   - BME280 I2C (Adafruit/compatibles) -> SDA=21, SCL=22 (par défaut ESP32)
   - Sonde d'humidité du sol analogique (sortie analogique) -> pin 34 (entrée uniquement)
   - Relais pour électrovanne -> pin 26 (active LOW ou HIGH selon module, voir RELAY_ACTIVE_LEVEL)
   - Bouton poussoir -> pin 14 vers GND (INPUT_PULLUP)
   - Module audio série « type YX5300/DFPlayer Mini » (souvent appelé HW-*** sur la carte)
      * TX du module -> pin 16 (RX2 de l'ESP32)
      * RX du module -> pin 17 (TX2 de l'ESP32)
      * GND commun, Vcc 5V (et relier GND du 5V au GND ESP32). Sortie audio vers ampli/HP.

  Librairies Arduino à installer (Gestionnaire de bibliothèques):
   - Adafruit BME280 Library (installe aussi Adafruit Unified Sensor / BusIO)
   - ArduinoJson (>= 6.x)

  Note sur le module audio: le nom « HW-311A » est ambigu sur Internet. Le code ci-dessous
```

```
 parle le protocole série YX5200/YX5300/DFPlayer (très répandu). Si votre module est différent,
 dites-le et adaptez les commandes.
*/

#include <WiFi.h>
#include <WebServer.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>
#include <Wire.h>
#include <Adafruit_BME280.h>

/************** CONFIG UTILISATEUR **************/
// -- Wi-Fi --
const char* WIFI_SSID = "inserer ssid";
const char* WIFI_PASS = "insérer MDPp";

// -- Open-Meteo (Météo-France) --
// Pas de clé API nécessaire. Utilise les modèles AROME/ARPEGE via Open-Meteo.
const double OWM_LAT = 47.31706931733405; // Exemple: Paris
const double OWM_LON = 5.11960451880114;

// Fréquence de rafraîchissement météo (ms)
const unsigned long OWM_REFRESH_MS = 2UL * 60UL * 60UL * 1000UL; // 2 heures

// Seuils par défaut (modifiables via la page Web)
float SOIL_DRY_PCT_THRESH = 35.0;    // En-dessous = sol jugé sec => arrosage (si pas de pluie prévue)
```

```
float RAIN_POP_THRESH    = 0.5;    // Probabilité de précipitation [0..1] au-dessus => on évite d'arroser

float RAIN_MM_THRESH     = 1.0;    // Somme de pluie prévue (prochaines 12h) en mm au-dessus => on évite d'arroser

unsigned long WATERING_DURATION_S = 300; // Durée d'arrosage automatique par cycle (secondes)


// Calibration de la sonde sol (à ajuster après mesure à vide et dans l'eau)

int SOIL_ADC_WET  = 1200; // lecture ADC quand très humide (valeur basse sur bcp de sondes)

int SOIL_ADC_DRY  = 3300; // lecture ADC quand très sec (valeur haute)


// Relais

const int PIN_RELAY = 27;

const bool RELAY_ACTIVE_LEVEL = HIGH; // mettez HIGH si votre module est actif à l'état haut


// Bouton

const int PIN_BUTTON = 14; // vers GND


// Sonde sol (ADC)

const int PIN_SOIL = 34; // ADC1_CH6


// BME280 I2C

Adafruit_BME280 bme; // I2C par défaut (Wire, addr 0x76)


// MP3 série (YX5200/YX5300/DFPlayer)

HardwareSerial MP3(2); // UART2

const int MP3_RX = 17; // RX2 <- TX du module
```

```
const int MP3_TX = 16; // TX2 -> RX du module
uint16_t currentTrack = 1; // incrémenté à chaque appui
uint8_t  mp3Volume = 20;   // 0..30

// Web server
WebServer server(80);

/************** ETAT **************/
// Météo
float last_pop_max12h = 0.0;  // probabilité max sur 12h
float last_rain_sum12h = 0.0; // mm
unsigned long lastOWM = 0;

// Capteurs
float lastTemp = NAN, lastHum = NAN, lastPress = NAN;
int   lastSoilRaw = 0;  // 0..4095
float lastSoilPct = NAN; // 0..100 (0 = trempé)

// Arrosage
bool autoMode = true;  // géré par capteurs + météo
bool watering = false;
unsigned long waterStartMs = 0;
unsigned long manualLockUntil = 0; // empêche auto pendant X ms après un arrêt manuel
unsigned long MANUAL_LOCK_MS = 10UL * 60UL * 1000UL; // 10 min
unsigned long MAX_WATER_MS   = 30UL * 60UL * 1000UL; // 30 min sécurité

// Détection appui
```

```
int lastButtonState = HIGH;
unsigned long lastDebounce = 0;
const unsigned long DEBOUNCE_MS = 50;

/************** OUTILS **************/
void setRelay(bool on) {
  digitalWrite(PIN_RELAY, on ? RELAY_ACTIVE_LEVEL : !RELAY_ACTIVE_LEVEL);
}

bool relayState() {
  return (digitalRead(PIN_RELAY) == RELAY_ACTIVE_LEVEL);
}

int clampi(int v, int lo, int hi) { return v < lo ? lo : (v > hi ? hi : v); }
float clampf(float v, float lo, float hi) { return v < lo ? lo : (v > hi ? hi : v); }

float soilPctFromRaw(int raw) {
  // Map linéaire: raw_dry -> 0% humidité (sec), raw_wet -> 100% humidité
  raw = clampi(raw, min(SOIL_ADC_WET, SOIL_ADC_DRY), max(SOIL_ADC_WET,
SOIL_ADC_DRY));
  float pct = (float)(SOIL_ADC_DRY - raw) * 100.0 / (float)(SOIL_ADC_DRY - SOIL_ADC_WET);
  return clampf(pct, 0.0, 100.0);
}

/************** LOGGING SÉRIE **************/
void logStatus(const char* src) {
  unsigned long now = millis()/1000UL;
  bool btnPressed = (digitalRead(PIN_BUTTON) == LOW);
```

```
  Serial.printf("[STAT][%lus][%s] T=%.1f°C H=%.1f%% P=%.0fhPa | SoilRaw=%d
Soil=%.0f%% | POPmax12h=%.0f%% Rain12h=%.1fmm | Auto=%s Water=%s | Button=%s |
RSSI=%d dBm\n",
            now,
            src,
            isnan(lastTemp)?0:lastTemp,
            isnan(lastHum)?0:lastHum,
            isnan(lastPress)?0:lastPress,
            lastSoilRaw,
            isnan(lastSoilPct)?0:lastSoilPct,
            last_pop_max12h*100.0f,
            last_rain_sum12h,
            autoMode?"ON":"OFF",
            watering?"ON":"OFF",
            btnPressed?"PRESSED":"released",
            WiFi.RSSI());
}

void logWaterEvent(const char* msg) {
  Serial.printf("[WATER] %s | relay=%s | durAuto=%lus | max=%lums\n",
            msg,
            relayState()?"ON":"OFF",
            WATERING_DURATION_S,
            MAX_WATER_MS);
}

/************** MP3 (YX5200/YX5300/DFPlayer) **************/
// Trame: 0x7E FF 06 CMD 00 PH PL CHK_H CHK_L 0xEF
```

```
uint16_t mp3Checksum(uint8_t cmd, uint16_t param) {
  uint16_t sum = 0xFF + 0x06 + cmd + 0x00 + (param >> 8) + (param & 0xFF);
  return (uint16_t)(0 - sum);
}

void mp3Send(uint8_t cmd, uint16_t param) {
  uint16_t chk = mp3Checksum(cmd, param);
  uint8_t frame[10] = {0x7E, 0xFF, 0x06, cmd, 0x00, (uint8_t)(param >> 8), (uint8_t)(param & 0xFF), (uint8_t)(chk >> 8), (uint8_t)(chk & 0xFF), 0xEF};
  MP3.write(frame, 10);
}

void mp3SetVolume(uint8_t vol) { // 0..30
  vol = (vol > 30) ? 30 : vol;
  mp3Send(0x06, vol);
}

void mp3PlayTrack(uint16_t idx) { // jouer piste n (racine, nommées 0001.mp3, 0002.mp3, ...)
  mp3Send(0x03, idx);
}

void mp3Next() { mp3Send(0x01, 0); }
void mp3Prev() { mp3Send(0x02, 0); }
void mp3Stop() { mp3Send(0x16, 0); }

/************** METHODE CAPTEURS **************/
bool readBME() {
```

```
  lastTemp = bme.readTemperature();
  lastHum  = bme.readHumidity();
  lastPress = bme.readPressure() / 100.0; // hPa
  return !(isnan(lastTemp) || isnan(lastHum) || isnan(lastPress));
}

void readSoil() {
  lastSoilRaw = analogRead(PIN_SOIL);
  lastSoilPct = soilPctFromRaw(lastSoilRaw);
}

/************** METEO OPENWEATHER **************/
bool fetchWeather() {
  if (WiFi.status() != WL_CONNECTED) {
    Serial.println("[WEATHER] Wi-Fi non connecté, impossible de récupérer la météo.");
    return false;
  }
  HTTPClient http;
  String url = String("https://api.open-meteo.com/v1/meteofrance?latitude=") +
String(OWM_LAT, 6) +
          "&longitude=" + String(OWM_LON, 6) +
          "&hourly=precipitation_probability,precipitation"
          "&forecast_hours=12"
          "&timezone=auto"; // auto = fuseau local
  http.begin(url);
  int code = http.GET();
  if (code != 200) { http.end(); Serial.printf("[WEATHER] HTTP %d\n", code); return false; }
```

```
  DynamicJsonDocument doc(16384);
  DeserializationError err = deserializeJson(doc, http.getStream());
  http.end();
  if (err) { Serial.printf("[WEATHER] JSON error: %s\n", err.c_str()); return false; }

  JsonArray pop = doc["hourly"]["precipitation_probability"].as<JsonArray>();
  JsonArray mm  = doc["hourly"]["precipitation"].as<JsonArray>();
  if (pop.isNull() || mm.isNull() || pop.size() == 0) { Serial.println("[WEATHER] Données horaires absentes."); return false; }

  float popMax = 0.0f, rainSum = 0.0f;
  for (size_t i = 0; i < pop.size(); ++i) {
    float p = pop[i].isNull() ? 0.0f : ((float)pop[i] / 100.0f); // % -> 0..1
    float r = mm[i].isNull() ? 0.0f : (float)mm[i];            // mm/h
    if (p > popMax) popMax = p;
    rainSum += r;
  }

  last_pop_max12h = popMax;
  last_rain_sum12h = rainSum;
  lastOWM = millis();
  Serial.printf("[WEATHER] POPmax12h=%.0f%% | Rain12h=%.1fmm\n", popMax*100.0f, rainSum);
  return true;
}

/************** LOGIQUE D'ARROSAGE **************/
void controlLoop() {
```

```
// rafraîchir capteurs
readSoil();
readBME();

// météo périodique
if (millis() - lastOWM > OWM_REFRESH_MS) {
 fetchWeather();
}

// gestion arrosage
unsigned long now = millis();

// fin automatique si temps dépassé
if (watering && (now - waterStartMs > MAX_WATER_MS)) {
 watering = false;
 setRelay(false);
 logWaterEvent("Arrêt sécurité (MAX_WATER_MS dépassé)");
}

if (!autoMode) {
 logStatus("MANUAL");
 return; // en manuel, on ne déclenche rien tout seul
}

// si verrou manuel récent, ne rien faire
if (now < manualLockUntil) {
 logStatus("LOCK");
```

```
    return;
  }

  // Décision arrosage
  bool soilDry = (lastSoilPct < SOIL_DRY_PCT_THRESH);
  bool rainSoon = (last_pop_max12h >= RAIN_POP_THRESH) || (last_rain_sum12h >=
RAIN_MM_THRESH);

  if (!watering) {
    if (soilDry && !rainSoon) {
      // démarrer un cycle
      watering = true;
      waterStartMs = now;
      setRelay(true);
      logWaterEvent("DÉMARRAGE arrosage (auto)");
    }
  } else {
    // en cours: arrêter quand la durée est atteinte
    if (now - waterStartMs >= WATERING_DURATION_S * 1000UL) {
      watering = false;
      setRelay(false);
      logWaterEvent("ARRÊT arrosage (durée atteinte)");
    }
  }

  logStatus("AUTO");
}
```

```
/************** PAGE WEB **************/

const char* INDEX_HTML = R"HTML(

<!doctype html>

<html lang="fr">

<head>

 <meta charset="utf-8" />

 <meta name="viewport" content="width=device-width, initial-scale=1" />

 <title>Arrosage ESP32</title>

 <style>

  body{font-family:system-ui,Segoe UI,Roboto,Arial,sans-
serif;margin:0;padding:2rem;background:#0e1117;color:#e6edf3}

  .card{background:#161b22;border:1px solid #30363d;border-
radius:12px;padding:1rem;margin:0 0 1rem}

  .row{display:flex;gap:1rem;flex-wrap:wrap}

  .row>.card{flex:1 1 320px}

  label{display:block;margin:.4rem 0 .2rem;color:#a9b1ba}

  input[type=number]{width:100%;padding:.5rem;border-radius:8px;border:1px solid
#30363d;background:#0e1117;color:#e6edf3}

  button{border:1px solid #30363d;background:#1f6feb;color:white;border-
radius:10px;padding:.6rem 1rem;cursor:pointer}

  button.secondary{background:#30363d}

  .grid{display:grid;grid-template-columns:repeat(auto-fit,minmax(220px,1fr));gap:1rem}

  h1{margin-top:0}

  .ok{color:#56d364}.warn{color:#f0883e}.bad{color:#ffa198}

  small{color:#8b949e}

  table{width:100%;border-collapse:collapse}

  td{padding:.3rem .2rem;border-bottom:1px solid #30363d}

 </style>

</head>
```

```
<body>
 <h1> 🌿 Arrosage connecté</h1>
 <div class="row">
  <div class="card">
   <h3>Capteurs</h3>
   <table>
    <tr><td>Température</td><td id="t">—</td></tr>
    <tr><td>Humidité air</td><td id="h">—</td></tr>
    <tr><td>Pression</td><td id="p">—</td></tr>
    <tr><td>Sol (brut)</td><td id="sr">—</td></tr>
    <tr><td>Sol (%)</td><td id="sp">—</td></tr>
    <tr><td>Wi-Fi RSSI</td><td id="rssi">—</td></tr>
   </table>
  </div>
  <div class="card">
   <h3>Météo (12h)</h3>
   <table>
    <tr><td>Probabilité pluie max</td><td id="pop">—</td></tr>
    <tr><td>Pluie cumulée</td><td id="rain">—</td></tr>
   </table>
   <div style="margin-top:.6rem">
    <button class="secondary" onclick="refreshForecast()">↻ Rafraîchir</button>
   </div>
   <small>Source: Open-Meteo / Météo-France</small>
  </div>
  <div class="card">
   <h3>Arrosage</h3>
```

```
  <div id="state">—</div>
  <div style="margin-top:.6rem;display:flex;gap:.5rem;flex-wrap:wrap">
   <button onclick="manual('on')">Démarrer (manuel)</button>
   <button class="secondary" onclick="manual('off')">Arrêter</button>
   <button class="secondary" onclick="toggleMode()" id="modebtn">Mode…</button>
  </div>
 </div>
 <div class="card">
  <h3>Audio</h3>
  <div class="grid">
   <button onclick="play(1)">▶ Piste 1</button>
   <button onclick="play(2)">▶ Piste 2</button>
   <button onclick="play(3)">▶ Piste 3</button>
   <button onclick="play(0)">■ Stop</button>
  </div>
 </div>
</div>
<div class="card">
 <h3>Réglages</h3>
 <div class="grid">
  <div>
   <label>Seuil sol sec (%)</label>
   <input id="th_soil" type="number" min="0" max="100" step="1">
  </div>
  <div>
   <label>Seuil prob. pluie (0..1)</label>
   <input id="th_pop" type="number" min="0" max="1" step="0.05">
```

```
  </div>
  <div>
   <label>Seuil pluie cumulée (mm/12h)</label>
   <input id="th_mm" type="number" min="0" max="50" step="0.5">
  </div>
  <div>
   <label>Durée arrosage auto (s)</label>
   <input id="dur" type="number" min="30" max="3600" step="10">
  </div>
 </div>
 <div style="margin-top:.6rem"><button onclick="save()">💾 Enregistrer</button></div>
</div>

<script>
 async function getData(){
  const r = await fetch('/api/data');
  const j = await r.json();
  document.getElementById('t').textContent = j.temp.toFixed(1)+' °C';
  document.getElementById('h').textContent = j.hum.toFixed(1)+' %';
  document.getElementById('p').textContent = j.press.toFixed(0)+' hPa';
  document.getElementById('sr').textContent = j.soilRaw;
  document.getElementById('sp').textContent = j.soilPct.toFixed(0)+' %';
  document.getElementById('rssi').textContent = j.rssi+' dBm';
  document.getElementById('pop').textContent = (j.popMax12h*100).toFixed(0)+' %';
  document.getElementById('rain').textContent = j.rainSum12h.toFixed(1)+' mm';
  document.getElementById('th_soil').value = j.th_soil;
  document.getElementById('th_pop').value = j.th_pop;
```

```
    document.getElementById('th_mm').value = j.th_mm;
    document.getElementById('dur').value = j.dur;
    document.getElementById('modebtn').textContent = j.autoMode? 'Passer en MANUEL' : 'Passer en AUTO';
    const s = j.watering? '💧 Arrosage EN COURS' : '⏸ Arrosage à l\'arrêt';
    document.getElementById('state').innerHTML = s + '<br><small>'+j.decision+'</small>';
  }

  async function save(){
    const body = new URLSearchParams();
    body.set('th_soil', document.getElementById('th_soil').value);
    body.set('th_pop', document.getElementById('th_pop').value);
    body.set('th_mm', document.getElementById('th_mm').value);
    body.set('dur', document.getElementById('dur').value);
    await fetch('/api/config', {method:'POST', headers:{'Content-Type':'application/x-www-form-urlencoded'}, body});
    getData();
  }

  async function manual(cmd){
    await fetch('/api/manual', {method:'POST', headers:{'Content-Type':'application/x-www-form-urlencoded'}, body:'water='+cmd});
    getData();
  }

  async function toggleMode(){
    await fetch('/api/config', {method:'POST', headers:{'Content-Type':'application/x-www-form-urlencoded'}, body:'toggleMode=1'});
    getData();
```

```
  }

  async function play(n){
    await fetch('/api/audio', {method:'POST', headers:{'Content-Type':'application/x-www-form-urlencoded'}, body:'track='+n});
  }

  async function refreshForecast(){
    await fetch('/api/refreshForecast', {method:'POST'});
    getData();
  }

  getData();
  setInterval(getData, 3000);
 </script>
</body>
</html>
)HTML";

/************** HANDLERS HTTP **************/
String decisionString(bool soilDry, bool rainSoon) {
 String s = String("Sol ") + (soilDry? "sec" : "humide") + ", pluie prévue: " + (rainSoon? "OUI" : "non");
 s += String(". Seuil sol=") + SOIL_DRY_PCT_THRESH + "% , pop>= " + RAIN_POP_THRESH + ", mm>= " + RAIN_MM_THRESH;
 return s;
}

void handleIndex(){
```

```
  server.send(200, "text/html", INDEX_HTML);
}

void handleData(){
  // snapshot
  bool soilDry = (lastSoilPct < SOIL_DRY_PCT_THRESH);
  bool rainSoon = (last_pop_max12h >= RAIN_POP_THRESH) || (last_rain_sum12h >= RAIN_MM_THRESH);

  StaticJsonDocument<1024> doc;
  doc["temp"] = isnan(lastTemp)? 0 : lastTemp;
  doc["hum"]  = isnan(lastHum)? 0 : lastHum;
  doc["press"] = isnan(lastPress)? 0 : lastPress;
  doc["soilRaw"] = lastSoilRaw;
  doc["soilPct"] = isnan(lastSoilPct)? 0 : lastSoilPct;
  doc["popMax12h"] = last_pop_max12h;
  doc["rainSum12h"] = last_rain_sum12h;
  doc["autoMode"] = autoMode;
  doc["watering"] = watering;
  doc["th_soil"] = SOIL_DRY_PCT_THRESH;
  doc["th_pop"]  = RAIN_POP_THRESH;
  doc["th_mm"]   = RAIN_MM_THRESH;
  doc["dur"]     = WATERING_DURATION_S;
  doc["rssi"] = WiFi.RSSI();
  doc["decision"] = decisionString(soilDry, rainSoon);

  String out; serializeJson(doc, out);
  server.send(200, "application/json", out);
```

```
}

void handleConfig(){
 if (server.hasArg("toggleMode")) {
  autoMode = !autoMode;
  Serial.printf("[CFG] Mode %s\n", autoMode?"AUTO":"MANUEL");
 }
 if (server.hasArg("th_soil")) {
  SOIL_DRY_PCT_THRESH = clampf(server.arg("th_soil").toFloat(), 0, 100);
  Serial.printf("[CFG] th_soil=%.1f%%\n", SOIL_DRY_PCT_THRESH);
 }
 if (server.hasArg("th_pop")) {
  RAIN_POP_THRESH = clampf(server.arg("th_pop").toFloat(), 0, 1);
  Serial.printf("[CFG] th_pop=%.2f\n", RAIN_POP_THRESH);
 }
 if (server.hasArg("th_mm")) {
  RAIN_MM_THRESH = max(0.0f, server.arg("th_mm").toFloat());
  Serial.printf("[CFG] th_mm=%.1fmm\n", RAIN_MM_THRESH);
 }
 if (server.hasArg("dur")) {
  WATERING_DURATION_S = max(30UL, (unsigned long) server.arg("dur").toInt());
  Serial.printf("[CFG] dur=%lus\n", WATERING_DURATION_S);
 }
 server.send(200, "text/plain", "OK");
}

void handleManual(){
```

```
  if (server.hasArg("water")) {
    String cmd = server.arg("water");
    if (cmd == "on") {
      autoMode = false; // passe en manuel
      watering = true;
      waterStartMs = millis();
      setRelay(true);
      logWaterEvent("DÉMARRAGE manuel");
    } else if (cmd == "off") {
      watering = false;
      setRelay(false);
      manualLockUntil = millis() + MANUAL_LOCK_MS; // évite redémarrage auto immédiat
      logWaterEvent("ARRÊT manuel (+verrou 10min)");
    }
  }
  server.send(200, "text/plain", "OK");
}

void handleAudio(){
  if (!server.hasArg("track")) { server.send(400, "text/plain", "missing track"); return; }
  int t = server.arg("track").toInt();
  if (t == 0) { mp3Stop(); Serial.println("[AUDIO] Stop"); }
  else { mp3PlayTrack((uint16_t)t); Serial.printf("[AUDIO] Play %d\n", t); }
  server.send(200, "text/plain", "OK");
}

void handleRefreshForecast(){
```

```
  bool ok = fetchWeather();
  server.send(ok?200:500, "text/plain", ok?"OK":"ERR");
}

/************** SETUP / LOOP **************/
void setup() {
  Serial.begin(115200);
  delay(200);
  Serial.println("Booting...");

  pinMode(PIN_RELAY, OUTPUT);
  setRelay(false);

  pinMode(PIN_BUTTON, INPUT_PULLUP);

  // ADC
  analogReadResolution(12); // 0..4095

  // I2C + BME280
  Wire.begin();
  bool bmeOk = bme.begin(0x76);
  if (!bmeOk) {
    Serial.println("BME280 introuvable (addr 0x76). Essai 0x77...");
    bmeOk = bme.begin(0x77);
  }
  if (!bmeOk) Serial.println("[WARN] BME280 non détecté.");
```

```
  // MP3
  MP3.begin(9600, SERIAL_8N1, MP3_RX, MP3_TX);
  delay(200);
  mp3SetVolume(mp3Volume);

  // Wi-Fi
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  Serial.print("WiFi...");
  unsigned long t0 = millis();
  while (WiFi.status() != WL_CONNECTED && millis() - t0 < 20000) { delay(250);
Serial.print('.'); }
  Serial.println();
  if (WiFi.status() == WL_CONNECTED) {
    Serial.print("IP: "); Serial.println(WiFi.localIP());
  } else {
    Serial.println("[WARN] Non connecté Wi-Fi. La partie météo/page web nécessitera le
réseau.");
  }

  // HTTP routes
  server.on("/", handleIndex);
  server.on("/api/data", handleData);
  server.on("/api/config", HTTP_POST, handleConfig);
  server.on("/api/manual", HTTP_POST, handleManual);
  server.on("/api/audio", HTTP_POST, handleAudio);
  server.on("/api/refreshForecast", HTTP_POST, handleRefreshForecast);
  server.begin();
```

```
  // premier fetch météo
  fetchWeather();

  Serial.println("Setup OK. Ouvrez le Moniteur Série @115200 bauds.");
}

void loop() {
  server.handleClient();

  // bouton -> joue piste suivante
  int reading = digitalRead(PIN_BUTTON);
  if (reading != lastButtonState) {
    lastDebounce = millis();
    lastButtonState = reading;
  }
  if ((millis() - lastDebounce) > DEBOUNCE_MS) {
    static bool wasLow = false;
    bool isLow = (reading == LOW);
    if (isLow && !wasLow) {
      // front descendant -> action
      currentTrack++;
      if (currentTrack > 20) currentTrack = 1; // suppose <=20 pistes
      Serial.printf("[BTN] Press detected -> play track %u\n", currentTrack);
      mp3PlayTrack(currentTrack);
    }
    if (!isLow && wasLow) {
```

```
      Serial.println("[BTN] Release");
    }
    wasLow = isLow;
  }

  // boucle de contrôle toutes 2s
  static unsigned long lastCtrl = 0;
  if (millis() - lastCtrl > 2000) {
    lastCtrl = millis();
    controlLoop();
  }
}
```