Documentation Technique

Station Meteo ESP32

Version 3.0

Janvier 2026

Amin Torrisi - CPNV

Auteur: Amin Torrisi / Équipe CPNV

Date: Janvier 2026

Version: 3.0 (WiFi Manager + Portail Captif)

Table des Matieres

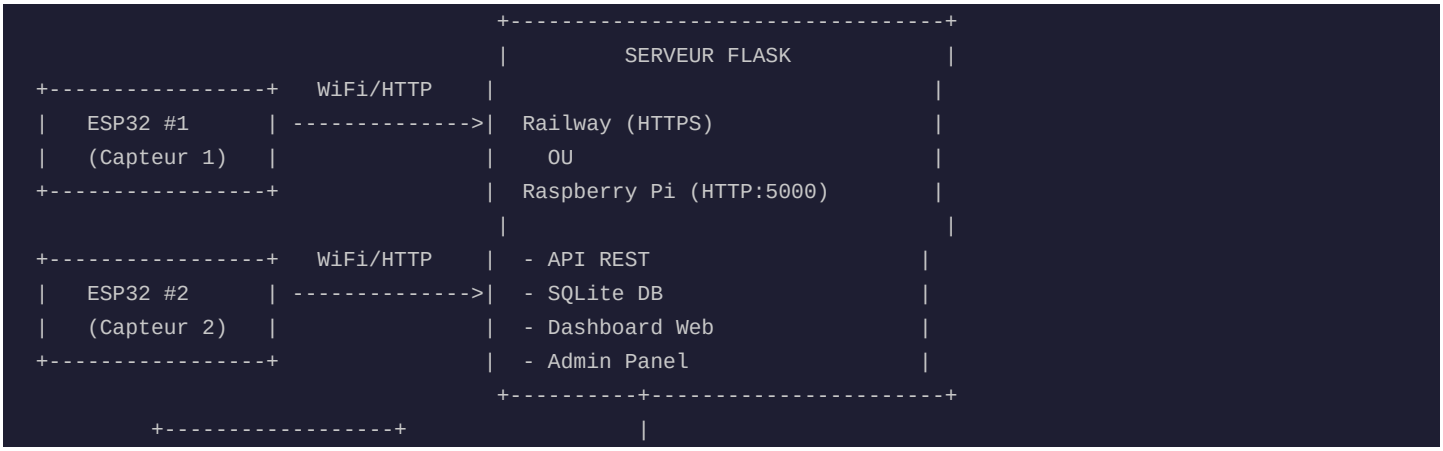
1. Vue d'ensemble

Description du projet

Station meteo connectee composee de :

- Capteurs ESP32 (Atom Lite M5Stack) avec capteurs de temperature, humidite et pression
- Serveur Flask deployable sur Railway (cloud) ou Raspberry Pi (local)
- Interface web pour visualiser les donnees en temps reel
- Interface admin pour gerer les ESP32 a distance
- WiFi Manager : portail captif pour configurer les ESP32 sans modifier le code

Schema global



```
          |   WiFi Manager   |                  ?
          | (Portail Captif) |         +------------------+
          | 192.168.4.1      |         |   Navigateur     |
          | Config WiFi +    |         |   (Dashboard)    |
          | Serveur IP       |         +------------------+
          +------------------+
```

2. Architecture du systeme

Stack technique

Composant	Technologie
Microcontroleur	ESP32 Atom Lite (M5Stack)
Capteur Temp/Humidite	SHT40 (ENV IV)
Capteur Pression	BMP280 (ENV IV)
Backend	Python Flask
Base de donnees	SQLite
Frontend	HTML / CSS / JavaScript / Chart.js
Hebergement Cloud	Railway.app (HTTPS)
Hebergement Local	Raspberry Pi (HTTP:5000)
WiFi Manager	WebServer + DNSServer + Preferences (NVS)

Communication

```
ESP32 --> POST /request/                --> Flask --> SQLite (espX)
         (JSON: capteur_id, mac, temp, hum, pression)

ESP32 --> POST /api/esp32/register      --> Flask --> esp32_devices
         (JSON: mac_address, ip_address)

ESP32 --> GET /api/esp32/config/{mac}   --> Flask --> sensor_number + capteur_id
```

3. Composants materiels

ESP32 Atom Lite (M5Stack)

```
+-------------------------------------+
|         ATOM LITE                   |
|    +---------------------+          |
|    |      LED (GPIO 27)  |          |
|    |      BTN (GPIO 39)  |          |
|    +---------------------+          |
|                                     |
|    Grove Port (I2C):                |
```

```
|    - SDA: GPIO 26                |
|    - SCL: GPIO 32                |
|    - 5V, GND                     |
+----------------------------------+
         | I2C Bus
         ?
+----------------------------------+
|         ENV IV Unit              |
|    +---------+  +---------+      |
|    | SHT40   |  | BMP280  |      |
|    | 0x44    |  | 0x76/77 |      |
|    | Temp+Hum|  | Pression|      |
|    +---------+  +---------+      |
+----------------------------------+
```

Pinout

Fonction	GPIO
Bouton	39 (INPUT_ONLY, pas de pullup interne)
LED	27
I2C SDA	26
I2C SCL	32

4. Structure des fichiers

```
Station meteo/
+-- app/
|   +-- __init__.py
|   +-- main.py              # Point d'entree Flask
|   +-- route.py             # Routes pages (/, /admin, /about...)
|   +-- api.py               # API REST (capteurs + ESP32)
|   +-- esp.py               # Reception donnees ESP32 (/request/)
|   +-- database.py          # Fonctions SQLite
|   +-- templates/
|   |   +-- index.html       # Dashboard principal
|   |   +-- admin.html       # Administration ESP32
|   |   +-- statistical.html # Graphiques (Chart.js)
|   |   +-- history.html     # Historique des mesures
|   |   +-- about.html       # Page a propos
|   +-- static/
|       +-- style.css        # Styles CSS (theme sombre)
|       +-- icone.png        # Favicon
|       +-- js/
|           +-- data.js      # JS Dashboard (cartes capteurs)
|           +-- admin.js     # JS Admin (config ESP32)
|           +-- statistical.js # JS Graphiques (Chart.js)
|           +-- history.js   # JS Historique (tableau)
|           +-- carte.js     # JS Carte/resume
|
+-- code esp32/
|   +-- capteur_auto/
```

```
|   |   +-- capteur_auto.ino # Code principal (WiFi Manager)
|   +-- capteur_1/           # Ancien code capteur 1 (deprecie)
|   +-- capteur_2/           # Ancien code capteur 2 (deprecie)
|   +-- capteur_universel/   # Code universel (deprecie)
|   +-- secrets.h            # Template credentials
|
+-- doc/                     # Documentation interne
+-- scripts/                 # Scripts de demarrage local
+-- Procfile                 # Config Railway
+-- requirements.txt         # Dependances Python
+-- weather_data.db          # Base SQLite (auto-creee)
+-- README.md
```

Note : Le fichier capteur_auto.ino est le seul sketch a utiliser. Les autres (capteur_1, capteur_2, capteur_universel) sont conserves comme reference mais sont deprecies.

5. Base de donnees

Schema SQLite

Tables esp1, esp2, ... (donnees capteurs)

```
CREATE TABLE espX (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    temperature REAL,        -- Peut etre NULL (sync inter-capteurs)
    humidity REAL,           -- Peut etre NULL
    pressure REAL,           -- Peut etre NULL
    date TEXT NOT NULL,      -- Format: "2026-01-29"
    hour TEXT NOT NULL       -- Format: "14:30:45"
);
```

Important : Quand un capteur envoie des donnees, une ligne NULL est inseree dans la table de l'autre capteur pour maintenir la synchronisation temporelle. Le systeme de statut ignore ces lignes NULL.

Table esp32_devices (configuration ESP32)

```
CREATE TABLE esp32_devices (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    mac_address TEXT UNIQUE NOT NULL,
    sensor_number INTEGER,
    name TEXT,
    last_seen TEXT,
    ip_address TEXT,
    created_at TEXT DEFAULT CURRENT_TIMESTAMP
);
```

Relations

```
esp32_devices                    Tables donnees
+--------------------+          +--------------+
| mac: 94:B9:7E:...  |          |     esp1     |
| sensor_number: 1   |--ATOM_001-| temperature  |
| name: "Salon"      |          | humidity     |
| last_seen: ...     |          | pressure     |
+--------------------+          | date, hour   |
                                 +--------------+
```

6. Flux de donnees

Cycle complet d'un ESP32

```
1. PREMIER DEMARRAGE
   ESP32 cree AP "StationMeteo-XXXX" --> Portail captif (192.168.4.1)
   --> Config WiFi + IP serveur --> Sauvegarde NVS --> Redemarrage

2. DEMARRAGES SUIVANTS
   ESP32 lit config NVS --> Connexion WiFi
   --> POST /api/esp32/register (MAC + IP)
   --> GET /api/esp32/config/{mac} (recupere sensor_number)

3. BOUCLE PRINCIPALE (toutes les 20 sec)
   Lecture capteurs --> POST /request/ (temp, hum, pression)

4. RECONFIGURATION
   Appui 3 sec bouton --> Efface NVS --> Redemarrage en mode AP
```

Format JSON envoye par l'ESP32

```
{
    "capteur_id": "ATOM_001",
    "mac_address": "94:B9:7E:A9:1C:38",
    "temperature": 23.45,
    "humidite": 45.67,
    "pression": 1013.25
}
```

Mapping capteur_id vers table

capteur_id	Table SQLite
ATOM_001	esp1
ATOM_002	esp2
ATOM_XXX	espXXX

7. API Endpoints

Donnees des capteurs

Methode	Endpoint	Description
GET	/api/sensors	Liste tous les capteurs
GET	/api/sensors/status	Statut en ligne/hors ligne
GET	/api/sensor/{id}/latest	Dernieres valeurs d'un capteur
GET	/api/sensor/{id}/history?date=YYYY-MM-DD	Historique par date
GET	/api/all/latest	Dernieres valeurs de tous les capteurs
POST	/request/	Reception donnees ESP32

Gestion des ESP32

Methode	Endpoint	Description
POST	/api/esp32/register	Enregistre un nouvel ESP32
GET	/api/esp32/config/{mac}	Recupere la config d'un ESP32
GET	/api/esp32/devices	Liste tous les ESP32 enregistres
POST	/api/esp32/configure	Assigne numero + nom a un ESP32
POST	/api/esp32/delete	Supprime un ESP32

Autres

Methode	Endpoint	Description
GET	/api/dates_unique	Dates disponibles dans la DB

Exemples de requetes

```
# Statut des capteurs
curl http://IP:5000/api/sensors/status

# Dernieres valeurs capteur 1
curl http://IP:5000/api/sensor/1/latest

# Enregistrer un ESP32
curl -X POST http://IP:5000/api/esp32/register \
  -H "Content-Type: application/json" \
  -d '{"mac_address": "94:B9:7E:A9:1C:38"}'
```

8. Code ESP32 - WiFi Manager

Fichier : code esp32/capteur_auto/capteur_auto.ino

Librairies requises

Librairie	Usage
WiFi.h	Connexion WiFi
WiFiClientSecure.h	HTTPS (Railway)
HTTPClient.h	Requetes HTTP
WebServer.h	Serveur web portail captif
DNSServer.h	Redirection DNS captive
Preferences.h	Sauvegarde persistante NVS
Wire.h	Bus I2C
Adafruit_SHT4x.h	Capteur temperature/humidite
Adafruit_BMP280.h	Capteur pression
ArduinoJson.h	Parsing JSON

Les librairies WebServer, DNSServer, Preferences sont incluses avec le SDK ESP32. Installer uniquement : Adafruit SHT4x, Adafruit BMP280, ArduinoJson.

Flux de demarrage

```
Boot ESP32
    |
    ?
Lecture config NVS (Preferences)
    |
    +-- Pas de config --> Mode AP "StationMeteo-XXXX"
    |                     |
    |                     ?
    |                     Portail captif (192.168.4.1)
    |                     - Scan reseaux WiFi
    |                     - Formulaire: SSID + MDP + IP serveur
    |                     - Sauvegarde NVS -> Redemarrage
    |
    +-- Config trouvee --> Connexion WiFi sauvegarde
                           |
                           +-- Succes --> Enregistrement serveur
                           |              |
                           |              ?
                           |              Boucle principale
                           |              (lecture + envoi toutes les 20s)
                           |
                           +-- Échec --> Reset config -> Mode AP
```

Detection serveur local vs distant

```
bool isLocalServer() {
    // Si que des chiffres et points = IP locale (Raspberry Pi)
    // Si contient des lettres = URL (Railway)
}

String getServerURL() {
    if (isLocalServer()) return "http://IP:5000";
    else return "https://URL";
}
```

Bouton reset (3 secondes)

Maintenir le bouton GPIO 39 pendant 3 secondes :

- Efface toute la configuration NVS
- Redemarre l'ESP32 en mode portail captif
- Permet de changer de reseau WiFi ou de serveur

9. Interface Web

Pages

URL	Description	Fichier JS
/	Dashboard temps reel	data.js
/admin	Gestion ESP32	admin.js
/statistical	Graphiques Chart.js	statistical.js
/history	Tableau historique	history.js
/about	À propos	-

Dashboard (/)

- Cartes par capteur avec temperature, humidite, pression
- Indicateur de statut (vert/orange/rouge)
- Nom personnalise du capteur (defini dans admin)
- Actualisation automatique toutes les 20 secondes

Admin (/admin)

- Liste des ESP32 enregistres (MAC, IP, derniere connexion)
- Attribution numero de capteur + nom personnalise
- Suppression d'un ESP32
- Actualisation automatique

Graphiques (/statistical)

- Temperature (ligne)
- Humidite (barres)
- Pression (ligne)
- Selecteur de date
- Librairie Chart.js (CDN)

10. Statut des capteurs

Logique de determination

Le statut est calcule en comparant l'heure actuelle avec la derniere mesure non-NULL du capteur.

Statut	Couleur	Condition	Texte affiche
En ligne	Vert (pulsation)	< 2 minutes	"En ligne"
Recent	Orange	< 10 minutes	"Vu il y a X min"
Hors ligne	Rouge	> 10 minutes	"Hors ligne (X min/h/j)"
Jamais connecte	Rouge	Aucune donnee	"Jamais connecte"

Detail technique

- Requete SQL : SELECT date, hour FROM espX WHERE temperature IS NOT NULL ORDER BY id DESC LIMIT 1
- Les lignes avec temperature = NULL (inserees pour la synchronisation inter-capteurs) sont ignorees
- Fuseau horaire : Europe/Zurich
- Actualisation : toutes les 30 secondes cote frontend

CSS des indicateurs

```
.status-dot.online  { background: #4caf50; animation: pulse-green 2s infinite; }
.status-dot.recent  { background: #ff9800; }
.status-dot.offline { background: #f44336; }
```

11. Deploiement

Option 1 : Railway (Cloud)

```
URL : https://nurturing-achievement-production.up.railway.app
```

- Auto-deploy a chaque git push origin main
- HTTPS automatique
- Base SQLite persistante

```
git add .
git commit -m "Description"
git push origin main
```

Option 2 : Raspberry Pi (Local)

```
# Se connecter au Raspberry
ssh pi@192.168.1.XXX

# Aller dans le dossier
cd Station-meteo

# Installer les dependances
```

```
pip install -r requirements.txt

# Lancer le serveur
python main.py
# ou avec gunicorn :
gunicorn main:app -b 0.0.0.0:5000
```

Important : Utiliser 0.0.0.0 pour accepter les connexions externes (pas 127.0.0.1).

Configuration ESP32 selon le serveur

Serveur	Adresse a entrer dans le portail
Railway	nurturing-achievement-production.up.railway.app
Raspberry Pi	192.168.1.XXX (IP du Pi sur le reseau)

12. Problemes resolus

Bug: Crash ESP32 "Guru Meditation Error"

Cause : Les objets WiFiClient/WiFiClientSecure etaient declares dans des blocs if/else et detruits avant que HTTPClient n'ait fini de les utiliser.

Solution : Declarer les clients au niveau de la fonction.

Bug: Donnees NULL considerees comme activite

Cause : La requete get_sensor_last_activity recuperait la derniere ligne meme si toutes les valeurs etaient NULL.

Solution : Ajout de WHERE temperature IS NOT NULL dans la requete SQL.

Bug: Colonne ts inexistante

Cause : Le code inserait un champ ts dans la table de l'autre capteur, mais cette colonne n'existe pas.

Solution : Suppression du champ ts de l'insertion.

Bug: Noms capteurs non affiches

Cause : L'API /sensors/status ne renvoyait pas les noms personnalises depuis esp32_devices.

Solution : Ajout de la recuperation des noms et passage au frontend.

Bug: Chemins images casses sur Linux

Cause : Backslash \ dans les chemins src="static\icone.png".

Solution : Remplacement par des forward slashes /.

13. Annexes

Arduino IDE - Configuration

1. Carte : Outils -> Type de carte -> M5Stack-ATOM
2. Port : Selectionner le port COM de l'ESP32
3. Librairies a installer :

- Adafruit SHT4x

- Adafruit BMP280
- ArduinoJson

SQLite - Commandes utiles

```
-- Ouvrir la base
sqlite3 weather_data.db

-- Voir les tables
.tables

-- ESP32 enregistres
SELECT * FROM esp32_devices;

-- Dernieres mesures capteur 1
SELECT * FROM esp1 ORDER BY id DESC LIMIT 10;

-- Mesures avec valeurs reelles uniquement
SELECT * FROM esp1 WHERE temperature IS NOT NULL ORDER BY id DESC LIMIT 10;
```

Git - Commandes rapides

```
git status
git add .
git commit -m "Description"
git push origin main
```

Document redige par Amin Torrisi - Équipe CPNV - Janvier 2026

- GitHub : https://github.com/pj43svh/MA-Metier-Station-Meteo
- Trello : https://trello.com/b/mbKZJSjJ/station-meteo-esp32-rpi