Documentation Technique

Station Meteo ESP32

Auteur: Amin Torrisi / Equipe CPNV

Date: Janvier 2026

Version: 2.0 (avec interface admin)

Ce document contient la documentation technique complete du projet ainsi qu'un rapport detaille des modifications effectuees lors de la session de debug du 22 janvier 2026.

Table des Matieres

1. Vue d'ensemble

Qu'est-ce que ce projet ?

Une station meteo connectee composee de:
- 2 capteurs ESP32 (Atom Lite) avec capteurs de temperature, humidite et pression
- Un serveur Flask heberge sur Railway
- Une interface web pour visualiser les donnees en temps reel
- Une interface admin pour gerer les ESP32 a distance

Schema simplifie

```
ESP32 #1          WiFi/HTTPS        Serveur Flask
(Capteur 1)  ------------------>   (Railway)

ESP32 #2          WiFi/HTTPS        - API REST
(Capteur 2)  ------------------>   - SQLite DB
                                   - Dashboard
                                        |
                                        v
                                   Navigateur
                                   (Dashboard)
```

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
Hebergement	Railway.app
Graphiques	Matplotlib

Communication

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

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

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

3. Composants materiels

ESP32 Atom Lite (M5Stack)

L'ATOM Lite est un microcontroleur ESP32 compact de M5Stack.
Il dispose d'un port Grove pour connecter des capteurs I2C.

Broches Grove:
- SDA: GPIO 26
- SCL: GPIO 32
- Alimentation: 5V et GND

Capteurs ENV IV

Le module ENV IV contient deux capteurs:
- SHT40: Temperature et humidite (adresse I2C: 0x44)
- BMP280: Pression atmospherique (adresse I2C: 0x76 ou 0x77)

Capteur	Adresse I2C	Mesures
SHT40	0x44	Temperature, Humidite
BMP280	0x76/0x77	Pression

4. Structure des fichiers

```
Station meteo/
|
|-- main.py              # Point d'entree Flask
|-- route.py             # Routes principales
|-- api.py               # API REST + gestion ESP32
|-- esp.py               # Reception donnees ESP32
|-- database.py          # Fonctions SQLite
|-- statistical.py       # Generation graphiques
|-- requirements.txt     # Dependances Python
|
|-- templates/           # Pages HTML (Jinja2)
|   |-- index.html       # Dashboard principal
|   |-- admin.html       # Interface admin ESP32
|   |-- statistical.html # Page des graphiques
|   |-- history.html     # Historique des donnees
|   +-- about.html       # Page a propos
|
|-- static/              # Fichiers statiques
|   |-- style.css        # Styles CSS
|   |-- js/              # Scripts JavaScript
|   +-- graph_*.png      # Graphiques generes
|
+-- code esp32/
    +-- capteur_auto/
        +-- capteur_auto.ino  # Code Arduino
```

5. Base de donnees

Table espX (donnees des capteurs)

```
CREATE TABLE espX (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    temperature REAL,
    humidity REAL,
    pressure REAL,
    date TEXT NOT NULL,      -- Format: "2026-01-22"
    hour TEXT NOT NULL       -- Format: "14:30:45"
);
```

Table esp32_devices (configuration)

```
CREATE TABLE esp32_devices (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    mac_address TEXT UNIQUE NOT NULL,  -- "A4:CF:12:34:56:78"
    sensor_number INTEGER,              -- 1, 2, 3...
    name TEXT,                          -- "Capteur Salon"
    last_seen TEXT,                     -- Derniere connexion
    ip_address TEXT,                    -- IP locale ESP32
    created_at TEXT DEFAULT CURRENT_TIMESTAMP
);
```

6. Flux de donnees

1. Enregistrement d'un nouvel ESP32

Quand un ESP32 demarre pour la premiere fois:
1. Il se connecte au WiFi
2. Il recupere son adresse MAC (APRES connexion WiFi!)
3. POST /api/esp32/register avec {mac_address, ip_address}
4. Le serveur l'enregistre dans esp32_devices
5. Retourne "pending" si pas encore configure

2. Configuration via Admin

L'administrateur:
1. Accede a /admin
2. Voit les ESP32 en attente de configuration
3. Assigne un numero de capteur (1, 2, 3...)
4. Le serveur met a jour esp32_devices avec sensor_number

3. L'ESP32 recupere sa config

L'ESP32 verifie periodiquement sa configuration:
1. GET /api/esp32/config/{mac}
2. Si configure: recoit sensor_number et capteur_id
3. Stocke: sensorNumber = 1, capteurID = "ATOM_001"

4. Envoi des donnees

Toutes les 20 secondes:
1. POST /request/ avec {capteur_id, mac_address, temperature, humidite, pression}
2. Le serveur extrait le numero du capteur_id (ATOM_001 -> 1)
3. INSERT INTO esp1 (temperature, humidity, pressure, date, hour)

7. API Endpoints

Donnees des capteurs

Methode	Endpoint	Description
GET	/api/sensors	Liste tous les capteurs
GET	/api/sensors/status	Statut de tous les capteurs
GET	/api/sensor/{id}/latest	Dernieres valeurs d'un capteur
GET	/api/all/latest	Dernieres valeurs de tous
POST	/request/	Reception donnees ESP32

Gestion des ESP32

Methode	Endpoint	Description
POST	/api/esp32/register	Enregistre un ESP32
GET	/api/esp32/config/{mac}	Config d'un ESP32
GET	/api/esp32/devices	Liste tous les ESP32
POST	/api/esp32/configure	Configure un ESP32
POST	/api/esp32/delete	Supprime un ESP32

8. Code ESP32

Configuration WiFi

```
const char* WIFI_SSID = "Bomboclat";
const char* WIFI_PASSWORD = "zyxouzyxou";
const char* SERVER_BASE =
    "https://nurturing-achievement-production.up.railway.app";
```

Cycle de vie

DEMARRAGE:
1. Init I2C, Init SHT40, Init BMP280
2. Connexion WiFi
3. Recupere MAC (IMPORTANT: apres WiFi.begin()!)
4. POST /api/esp32/register {mac, ip}
5. GET /api/esp32/config/{mac} -> sensor_number

BOUCLE PRINCIPALE:
- Toutes les 20 sec: Lire capteurs, POST /request/
- Si pas configure: Toutes les 60 sec, verifier config
- Si bouton 3 sec: ESP.restart()

Format des donnees envoyees

```
{
  "capteur_id": "ATOM_001",
  "mac_address": "A4:CF:12:34:56:78",
  "temperature": 23.45,
  "humidite": 45.67,
  "pression": 1013.25
}
```

9. Interface Web

URL	Description
/	Dashboard - donnees en temps reel
/admin	Administration des ESP32
/statistical	Graphiques
/history	Historique des mesures
/about	A propos

Dashboard (/)

- Affiche une carte par capteur
- Donnees actualisees toutes les 5 secondes
- Indicateur online/offline

Admin (/admin)

- Liste tous les ESP32 enregistres
- Permet d'assigner un numero de capteur (1, 2, 3...)
- Affiche MAC, IP, derniere connexion
- Bouton pour supprimer un ESP32

10. Deploiement Railway

URL de production

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

Commande de deploiement

```
# Push vers GitHub (declenche auto-deploy)
git push origin interface-web-local:interface-web

# Si Railway ne prend pas le bon commit:
railway deploy --commit 76a7f2bb56f89a54ff585d7b71418cb7397b3a96
```

Fichier requirements.txt

```
flask==3.0.0
gunicorn==21.2.0
matplotlib==3.8.0
numpy==1.26.4
```

11. Problemes resolus

Bug #1: MAC Address 00:00:00:00:00:00

Probleme: Tous les ESP32 s'enregistraient avec la meme MAC invalide.

Cause: WiFi.macAddress() etait appele AVANT WiFi.begin().

Solution:

```
// AVANT (bug)
macAddress = WiFi.macAddress();  // Retourne 00:00:00:00:00:00
WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

// APRES (corrige)
WiFi.mode(WIFI_STA);
WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
// ... attendre connexion ...
macAddress = WiFi.macAddress();  // Retourne vraie MAC
```

Bug #2: Graphiques vides

Probleme: Les graphiques ne s'affichaient pas si esp1 etait vide.

Cause: L'axe des heures utilisait toujours esp1 en dur.

Solution: Utiliser dynamiquement le premier capteur disponible.

Bug #3: Double rafraichissement

Probleme: Les graphiques se generaient 2 fois a chaque clic.

Cause: refresh_statistical() appele 2 fois dans le JavaScript.

Solution: Supprime l'appel duplique.

RAPPORT DES MODIFICATIONS

Session du 22 Janvier 2026

Resume de la session

Cette session a consiste en un AUDIT COMPLET du site web de la station meteo, suivi de la correction de plusieurs bugs critiques et du deploiement des corrections.

Bugs identifies et corriges

Bug #1: Adresse MAC invalide

Fichier: code esp32/capteur_auto/capteur_auto.ino

Probleme: Tous les ESP32 s'enregistraient avec la meme adresse MAC 00:00:00:00:00:00.

Cause: WiFi.macAddress() etait appelee AVANT que le WiFi soit initialise. A ce moment, le module WiFi n'est pas encore actif et retourne une MAC par defaut.

Solution: Deplacer l'appel a WiFi.macAddress() APRES WiFi.begin() et attendre que la connexion soit etablie.

Bug #2: Double rafraichissement des graphiques

Fichier: static/js/statistical.js

Probleme: Les graphiques se generaient 2 fois a chaque clic sur "Rafraichir", causant un ralentissement et une double charge serveur.

Cause: La fonction refresh_statistical() etait appelee 2 fois dans le code.

Solution: Supprime l'appel duplique.

Bug #3: Graphiques vides

Fichier: api.py

Probleme: Les graphiques ne s'affichaient pas si la table esp1 etait vide, meme si esp2 contenait des donnees.

Cause: L'axe des heures etait recupere depuis esp1 en dur dans le code.

Solution: Utiliser dynamiquement le premier capteur disponible avec db.get_all_sensors()[0].

Bug #4: Crash sur donnees vides

Fichier: statistical.py

Probleme: La fonction create_graph_bar() plantait si aucune donnee n'etait trouvee pour la date selectionnee.

Cause: Pas de verification du retour de api_datas_list() avant utilisation.

Solution: Ajoute une verification "if not hour: return" au debut de la fonction.

Fichiers modifies (resume)

Fichier	Type de modification
capteur_auto.ino	Fix MAC address timing
static/js/statistical.js	Suppression appel duplique
api.py	Dynamisation axe des heures
statistical.py	Gestion erreurs donnees vides

Deploiement

Commit: 76a7f2bb56f89a54ff585d7b71418cb7397b3a96

Branche: interface-web-local -> interface-web

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

Ce qu'il faut retenir

1. WiFi.macAddress() doit etre appele APRES WiFi.begin()

Sinon le module WiFi n'est pas initialise et retourne 00:00:00:00:00:00

2. Les graphiques utilisent dynamiquement le premier capteur disponible

Cela evite les erreurs si esp1 est vide mais esp2 a des donnees

3. L'interface admin permet de configurer les ESP32 sans code

Aller sur /admin pour assigner un numero de capteur a chaque ESP32

4. Railway deploie automatiquement quand on push

git push origin interface-web-local:interface-web

Document genere automatiquement - Janvier 2026