Eduard GOLUB, Kolan ALLAIN, Noé LETEURTRE,
Simon PLAULT, Morgan VANVELTHEM

Solution technique Système d'Acquisition (SA)

Table des matières

Table des matières

Présentation technique Système d'Acquisition (SA)

1. Contexte du projet

L'objectif est de concevoir un système d'acquisition mobile permettant de mesurer en continu la température, l'humidité et le taux de CO_2 dans différentes salles d'un bâtiment universitaire.

Plusieurs systèmes d'acquisition (SA) identiques seront déployés, chacun pouvant être déplacé d'une salle à une autre selon les besoins.

Les données doivent être collectées automatiquement, transmises à un serveur central et consultables sur un site web.

Description générale du système

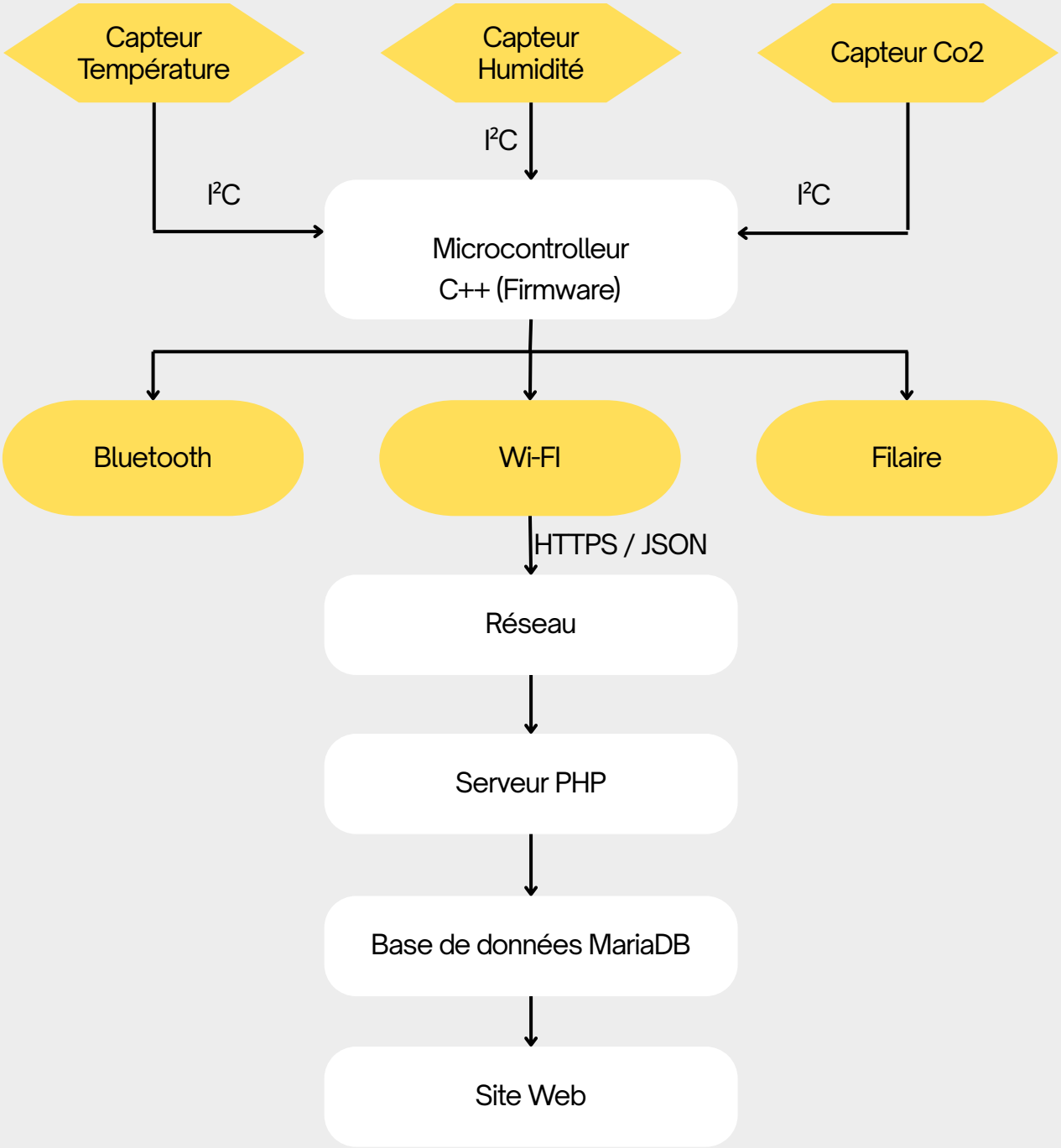
Description Générale du Système d'Acquisition (SA)

L'objectif est de concevoir un système de surveillance environnementale mobile et autonome capable de mesurer en continu la qualité de l'air (CO2,T,H) dans les salles d'un bâtiment universitaire.
Le Système d'Acquisition (SA) est modulaire. Sa conception repose sur la combinaison des pièces nécessaires suivantes :

Élément	Rôle Fonctionnel	Détails
Microcontrôleur	**Cerveau du système**	Gère l'acquisition des données, le traitement, l'affichage et la communication réseau.
Capteurs	**Acquisition des données**	Mesure des trois paramètres critiques : CO2 (dioxyde de carbone), Température (T) et Humidité (H).
Module de Communication	**Passerelle réseau**	Permet la transmission sans fil des données vers le serveur central (Wi-Fi retenu dans la solution finale).
Écran	**Interface Utilisateur Locale**	Fournit un affichage en temps réel des mesures pour validation immédiate.
Alimentation	**Autonomie énergétique**	Fournit l'énergie nécessaire au fonctionnement du SA (batterie rechargeable ou secteur).
Boîtier	**Protection physique**	Assure l'intégration et la protection des composants pour garantir la mobilité du SA.

Architecture fonctionnelle

Le fonctionnement global est illustré ci-dessous :



Chaque SA collecte les données toutes les 6 minutes, les affiche localement en direct, et les envoie périodiquement (6mn) vers le serveur sous format JSON via HTTP POST.

Cycle de Mesure et d'Envoi

Cycle d'Acquisition, d'Affichage Local et de Transmission BDD

Le SA utilise deux boucles distinctes pour optimiser la consommation : une boucle rapide pour l'affichage local et une boucle lente pour l'envoi à la BDD.

Étape du Cycle	Fréquence	Logique et Impact sur l'Autonomie
1. Acquisition des Mesures	Continu (Toutes les 30s)	Lecture des capteurs via I²C. Maintient le SA actif pour l'affichage local.
2. Affichage Local	En Direct	Mise à jour de l'écran TFT intégré avec les nouvelles valeurs, sans latence.
3. Déclenchement Envoi BDD	Toutes les 6 minutes	Le SA sort du mode économie d'énergie (ou interrompt le cycle) pour initier la connexion Wi-Fi et l'envoi.
4. Transmission Sécurisée	Une fois par cycle (ou après retry)	Envoi du paquet JSON (via HTTPS/API Token) et vérification du code de réponse du serveur.
5. Reprise du Cycle	Immédiate	Le SA désactive le Wi-Fi et revient à l'étape 1, continuant l'acquisition et l'affichage local. (Reste en mode Active).

Communication Microcontrôleur → Serveur

Voici comment les données sont transmises pour chaque solution :

Solution 1: Ethernet (Filaire)

- Matériel : Module externe LAN8720 (recommandé pour ESP32) ou W5500 (Shield SPI), connecté à l'ESP32.
- Protocole réseau : TCP/IP via câble RJ45.
- Méthode d'envoi :
  - Le module Ethernet établit une connexion TCP vers le serveur.
  - Requête HTTP POST envoyée au script PHP du serveur web.
  - Format des données : JSON.
- Avantages : Connexion très stable, pas d'interférences, sécurité réseau locale.

Solution 2: Wi-Fi (ESP32)

- Protocole réseau : TCP/IP via Wi-Fi 802.11n (2.4 GHz).
- Séquence de connexion :
  a. L'ESP32 se connecte au réseau Wi-Fi (SSID + mot de passe configurés).
  b. Il obtient une adresse IP automatiquement (DHCP).
  c. Il établit une connexion HTTP/HTTPS vers le serveur PHP.
- Méthode d'envoi :
  - Requête HTTP POST au format JSON vers le script PHP de l'API, incluant un API Token pour l'authentification
  - Le chiffrement HTTPS est obligatoire pour garantir la confidentialité des données (via TLS/SSL)
  - Reconnexion automatique en cas de perte de signal.
- Avantages : Installation sans câble, mobilité totale, utilise réseau existant.

Communication Microcontrôleur → Serveur

Solution 3: Bluetooth BLE

- Protocole réseau : Bluetooth Low Energy 4.2/5.0 (intégré à l'ESP32).
- Architecture en deux étapes :
  - Étape 1 - Capteur (ESP32) → Gateway (Bluetooth) :
    - L'ESP32 envoie ses données via BLE à une Passerelle centrale (Gateway).
    - Rôle de la Gateway : Elle est un dispositif (ex: Raspberry Pi) qui reçoit les données BLE à courte portée et les traduit en protocole Internet (HTTP/Wi-Fi) pour les envoyer au serveur.
    - Protocole : GATT (Generic Attribute Profile) utilisant des Services et des Caractéristiques.
    - Format : Données binaires compactes pour économiser la batterie.
  - Étape 2 - Gateway → Serveur PHP (Wi-Fi/Ethernet) :
    - La Gateway collecte les données de tous les capteurs BLE.
    - Elle les reformate en JSON et les envoie au serveur PHP via HTTP POST.
- Avantages : Très faible consommation, autonomie batterie de plusieurs mois.
- Inconvénient majeur : Nécessite une Gateway par zone (portée limitée à 10-15m) pour relayer les données vers Internet.

Analyse technique et choix

Solution	Avantages	Inconvénients	Recommandation
Filaire (Ethernet)	Très fiable et stable Sécurité élevée Connexion directe au serveur	Nécessite câblage RJ45 Peu mobile Installation plus lourde	Pour **installation permanente**
Wi-Fi (ESP32)	Sans fil, facile à déplacer Envoi automatique des données Coût raisonnable	Dépend du Wi-Fi local Sensible aux interférences Consommation moyenne	Pour **usage mobile et flexible**
Bluetooth (BLE)	Très faible consommation Coût réduit par unité	Portée courte Besoin d'une Gateway Architecture complexe	Pour **réseau dense (>20 capteurs)**

Justification Détaillée du Choix Wi-Fi

Pourquoi le Wi-Fi est la solution la plus viable
Le choix du Wi-Fi est stratégique pour l'économie d'infrastructure.

Avantage Clé	Impact sur le Projet
Coût d'Infrastructure	Zéro. Aucune Gateway additionnelle à acheter, installer ou maintenir (contrairement au BLE).
Couverture Totale	Le SA peut être déplacé dans l'intégralité du bâtiment sans perte de connexion.
Vitesse et Sécurité	Vitesse suffisante pour les envois périodiques. Supporte le HTTPS pour une sécurité optimale.

Robustesse, Sécurité et Tolérance de Panne

Mécanismes de Sécurité et de Fiabilité (Tolérance de Panne)

Le système est conçu pour protéger les données pendant le transport et garantir l'intégrité même en cas de panne réseau temporaire.

Domaine	Protocole ou Mécanisme	Rôle
Confidentialité	**HTTPS (TLS/SSL)**	Chiffrement du trafic HTTP entre le SA et le serveur.
Authentification	**API Token**	Jeton secret pour valider chaque SA auprès du serveur.
Fiabilité des Données	**Buffering en mémoire**	En cas d'échec de l'envoi Wi-Fi, les paquets sont mis en file d'attente et retransmis au cycle suivant.

Infrastructure et communication Reseau

1. Choix d’Eduroam comme support de communication L’infrastructure réseau de l’établissement (Eduroam) constitue le pilier de notre solution de communication. Ce choix d'exploiter le Wi-Fi universitaire est une décision stratégique qui maximise la mobilité tout en minimisant l’investissement matériel.

En utilisant le module Wi-Fi intégré nativement à l'ESP32, le projet ne nécessite aucun coût d'infrastructure additionnel :

- Contrairement au Bluetooth, aucune passerelle (Gateway) n'est requise.
- Contrairement à une solution filaire, aucun câblage Ethernet spécifique n'est nécessaire.

Cette approche garantit une couverture totale du bâtiment : le Système Autonome (SA) peut être déplacé sans interruption de service, profitant de la portée illimitée offerte par le déploiement existant du réseau sur le campus.

2. Fonctionnement technique avec l’ESP32 Le Wi-Fi offre un débit largement suffisant pour la transmission périodique de paquets JSON légers tout en supportant des protocoles robustes.

- Protocole réseau : TCP/IP via Wi-Fi 802.11n (2.4 GHz).
- Séquence de connexion :
    a. L’ESP32 s'authentifie sur le réseau Eduroam (SSID + identifiants configurés).
    b. Il obtient automatiquement une adresse IP via DHCP.
    c. Il établit une connexion sécurisée vers le serveur PHP.
- Méthode d’envoi : Les données sont transmises via des requêtes HTTP POST au format JSON. Le système gère une reconnexion automatique en cas de perte de signal, assurant la résilience du lien.

Infrastructure et communication Reseau

3. Sécurisation de la communication

La sécurité est implémentée à deux niveaux pour garantir l'intégrité et la confidentialité des données sur ce réseau institutionnel partagé :

- Sécurité Réseau (Transport) :
  - Authentification EAP-TLS/PEAP : Bien que complexe à implémenter sur microcontrôleur, cette méthode débloque l'accès à l'ensemble du réseau universitaire.
  - Chiffrement HTTPS : Le trafic est obligatoirement chiffré (TLS/SSL). Ceci garantit que les données ne peuvent être ni lues ni interceptées lors de leur transit sur le réseau.
- Sécurité Applicative :
  - Un Token d'API est inclus dans chaque requête POST pour restreindre l'accès au serveur aux seuls capteurs autorisés.

4. Validation de la connectivité

Afin de valider la faisabilité technique avant déploiement, nous avons réalisé des tests de connectivité (Ping vers une adresse externe type 8.8.8.8) et des tests de compatibilité TLS (via commande curl).

Ces tests ont permis de vérifier que le trafic sortant n'était pas bloqué et que le chiffrement était fonctionnel.

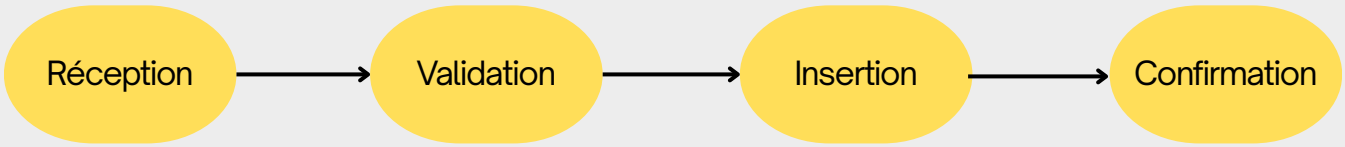
```
morgan@MacBook-Air-de-morgan ~ % ping -c 3 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: icmp_seq=0 ttl=116 time=12.525 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=116 time=15.656 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=116 time=12.878 ms

--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 12.525/13.686/15.656/1.400 ms
morgan@MacBook-Air-de-morgan ~ % ping -c 3 google.com
PING google.com (172.217.18.206): 56 data bytes
64 bytes from 172.217.18.206: icmp_seq=0 ttl=116 time=13.183 ms
64 bytes from 172.217.18.206: icmp_seq=1 ttl=116 time=11.863 ms
64 bytes from 172.217.18.206: icmp_seq=2 ttl=116 time=11.154 ms

--- google.com ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 11.154/12.067/13.183/0.841 ms
```

« Test de connectivité au réseau eduroam et compatibilité HTTPS, confirmant l'éligibilité pour l'envoi sécurisé de données JSON. »

Protocole de stockage : Microcontrôleur → Base de données

Réception des données côté serveur PHP

Le serveur web héberge un script PHP qui reçoit les requêtes HTTP POST et traite les données dans l'ordre suivant :



- Réception : Le script PHP reçoit le JSON (après décryptage TLS/SSL) via _POST ou php://input. Validation : Vérification de l'intégrité et de l'authenticité des données :
- Validation du format JSON (s'assurer que la structure est conforme aux attentes).
- Authentification du SA : Vérification de l'API Token (inclus dans l'en-tête de la requête) pour s'assurer que l'appareil est connu et autorisé.
- Vérification des plages de valeurs (s'assurer que T, H, CO2 sont dans des plages physiques raisonnables). Insertion SQL : Stockage des données validées dans la base de données MariaDB. Confirmation : Envoi d'un accusé de réception au capteur

Comparatif base de Données

Une base de données **simple** ne suffit pas pour un projet avec plusieurs tables, car elle ne permet pas de structurer correctement des données liées entre elles.

Une base de données **relationnelle** permet de modéliser proprement les relations, d'assurer l'intégrité des données grâce aux clés étrangères, et d'éviter les duplications.
Elle est donc beaucoup plus adaptée à notre projet structuré avec plusieurs entités connectée.

Critère	MariaDB	MySQL	PostgreSQL	SQLite
Type	Relationnelle (SQL)	Relationnelle (SQL)	Relationnelle avancée (SQL)	Relationnelle légère (SQL)
Licence	Open source (GPL)	Open source + version Oracle	Open source	Open source
Performances	Très bonnes, optimisé pour vitesse	Bonnes, parfois moins rapide depuis Oracle	Excellentes mais plus lourd	Très légères
Compatibilité	Totalement compatible MySQL	Référence historique	Très riche mais moins compatible MySQL	Fichiers locaux, pas de serveur
Évolutions	Rapides, communauté active	Dépend des décisions d'Oracle	Très actif	Stable mais limité
Réplication	Très performante, plus de modes	Standard mais moins flexible	Bonne	Limitée
Cas d'usage	Serveurs web, projets pro, évolutifs	Serveurs web traditionnels	Données complexes, contraintes fortes	Petites applis locales
Avantages clés	Rapide, libre, riche en fonctionnalités	Stable, connu, simple	Ultra robuste, très strict	Aucun serveur, hyper léger
Inconvénients	Quelques extensions spécifiques	Moins ouvert depuis Oracle	Plus complexe à gérer	Pas fait pour systèmes multi-utilisateurs

Solution Backend

Critère	PHP Natif (orienté objet)	Micro-framework PHP (Slim, Lumen)	PHP procédural
Architecture	PHP structuré, orienté objet, classes dédiées (API, sécurité, BDD)	PHP avec un framework léger, routage interne	Scripts PHP simples sans structure
Complexité	Moyenne	Faible à moyenne	Très faible
Performance	Excellente (aucune couche additionnelle)	Bonne (framework léger)	Maximale
Temps réel / API	API JSON simple, rafraîchissement côté client	API REST déjà intégrée via le framework	Pages statiques, API difficile
Maintenabilité	Bonne si structure respectée	Bonne (routage et organisation fournis)	Limitée, risque de désorganisation
Support IDE (PHPStorm)	Excellent (OOP, autocomplétion)	Excellent	Basique
Sécurité	Doit être implémentée manuellement (tokens, filtres)	Fonctions de sécurité intégrées	Très limitée
Déploiement sur Raspberry Pi	Simple (FTP/SSH + Nginx + PHP-FPM)	Simple mais nécessite installation framework	Très simple
Avantages	- Contrôle total du code - Performance optimale - Fonctionne parfaitement avec un Raspberry Pi - Débogage clair - Pas de dépendances externes	- Routage très propre - Structure déjà fournie - Extensible	- Développement immédiat - Idéal pour un prototype - Très faible charge
Inconvénients	- Routage à gérer soi-même - Sécurité à implémenter (token, validation)	- Fonctionnalités limitées vs grands frameworks - Chargement supplémentaire	- Risque de “code spaghetti” - Peu évolutif
Verdict	CHOIX OPTIMAL	BON compromis	RAPIDE pour MVP

Justification du choix retenu pour le projet

Pour le Système d'Acquisition que nous développons, la solution la plus adaptée est PHP natif orienté objet, pour plusieurs raisons directement liées aux besoins du projet :

1. Simplicité + Contrôle total
Le backend doit gérer uniquement :

- la réception des données JSON envoyées par l'ESP32,
- la validation et sécurisation du contenu,
- l'insertion dans MariaDB,
- la mise à disposition d'un endpoint consultable par l'interface web.

Ces opérations sont parfaitement réalisables sans framework, ce qui évite toute surcharge inutile.

2. Performances excellentes sur Raspberry Pi
Le Raspberry Pi héberge :

- Nginx,
- PHP-FPM,
- MariaDB.

PHP natif est extrêmement léger et rapide dans ce contexte, ce qui garantit une charge minimale et une disponibilité constante.

3. Structure claire, facile à maintenir
En utilisant une architecture orientée objet simple (classes : API, sécurité, base de données), on obtient un code :

- lisible,
- facile à étendre,
- compatible avec n'importe quel IDE moderne.

4. Sécurité maîtrisée
Le projet nécessite :

- une authentification par API Token,
- une validation stricte des champs reçus (CO_2, T, H, timestamp),
- une gestion d'erreurs claire.

PHP natif permet d'implémenter précisément les contrôles dont on a besoin, sans fonctionnalité superflue.

5. Déploiement très simple
Le code peut être déployé via :

- FTP,
- SCP/SSH,
- Git.

Aucun gestionnaire de dépendances ou environnement complexe n'est requis.

Solution Serveur Local : Raspberry Pi

Composant	Rôle dans l'architecture	Justification du choix
Matériel	Raspberry Pi (Modèle 3B, 4 ou 5)	Micro-ordinateur monocarte, faible consommation, suffisant pour accueillir simultanément l'API PHP natif et le site web Symfony.
Système d'exploitation	Raspberry Pi OS (Debian)	Système stable, léger, bien documenté, parfaitement compatible avec Nginx, PHP-FPM et MariaDB.
Serveur Web	Nginx	Gère les requêtes HTTP POST des ESP32 pour l'API et les requêtes HTTP du site Symfony. Nginx est idéal pour les environnements légers et performant sur Raspberry Pi.
Traitement – API capteurs	PHP natif	Code minimaliste et très performant pour recevoir, valider et insérer les données JSON. Réduit la charge du Raspberry Pi.
Traitement – Site Web	Symfony	Framework structuré, adapté pour l'interface utilisateur, la gestion des pages, le routing, la sécurité et l'affichage des données.
Base de données	MariaDB	Base de données relationnelle robuste utilisée pour stocker durablement les mesures T/H/CO_2 consultées par Symfony.

Solution Serveur Local : Raspberry Pi

Schéma de Fonctionnement

Le Raspberry Pi agit comme un stack LAMP/LEMP miniature (Linux, Apache/Nginx, MariaDB, PHP) :

1. Réception : L'ESP32 envoie le paquet JSON au serveur Web (Nginx) sur le Raspberry Pi.
2. Traitement : Nginx transmet la requête au moteur PHP qui exécute le script de validation.
3. Stockage : Le script PHP insère les données validées dans la base de données MariaDB.
4. Visualisation : Un site web (Page 8) hébergé sur le même Pi interroge MariaDB pour afficher les graphiques de données aux utilisateurs.

Avantages et Inconvénients

Avantages (Favorable au projet)	**Inconvénients (Points de vigilance)**
Coût minimal et autonomie	Maintenance locale (physique et logicielle).
Contrôle total des données et de l'environnement serveur.	**Fiabilité** dépendante de la connexion Internet locale et de l'alimentation.
Consommation électrique très faible.	**Sécurité** : Nécessite une configuration pare-feu et des mises à jour régulières pour protéger l'accès extérieur.
Évolutivité (via le Cloud)	Performances limitées par le matériel (moins adapté à un très grand nombre d'utilisateurs simultanés).

Solution Serveur Local : Raspberry Pi

Architecture Serveur (LEMP Stack)

Composant	Rôle dans l'architecture	Prix unitaire estimé (Investissement Initial)
Matériel	Raspberry Pi (Modèle 4 ou 5)	**~60 - 80$ €**
Stockage	Carte Micro SD (32 Go)	**~10 - 15$ €**
Alimentation	Alimentation USB-C officielle	**~8 - 12$ €**
Logiciel Serveur	Nginx, PHP, MariaDB	**0 €** (Logiciels Open Source)
TOTAL Coût d'Investissement (CAPEX)	**Mise en service unique**	**~78 - 107$ €**

Avantages et Coûts de Fonctionnement (OPEX)

Type de Coût	Détail	Coût de Fonctionnement Annuel (OPEX)
Énergie	Consommation électrique très faible (5-10 W en moyenne, 24/7).	**~10 - 25$ €/an** (selon le prix du kWh local).
Maintenance	Temps nécessaire pour les mises à jour logicielles, la gestion des pannes et la configuration réseau.	**Coût en temps** (variable, non monétaire)
Avantages Clés	**Contrôle total** des données, absence de frais d'abonnement mensuels, faible consommation.	
Inconvénients	Dépend de la fiabilité de l'alimentation locale et de la connexion Internet.	

Solution Cloud pour le Serveur

Solution Cloud	Technologie d'Ingestion	Avantages pour le Projet
AWS IoT Core	Protocole MQTT ou HTTP	Conçu spécifiquement pour l'IoT. Permet une ingestion sécurisée, scalable et bidirectionnelle.
Google Cloud IoT Core (ou Cloud Functions)	Protocole MQTT ou HTTP	Excellente intégration avec l'écosystème Google Cloud. Utilisation de fonctions sans serveur pour le traitement.
Azure IoT Hub	Protocole MQTT, AMQP ou HTTP	Offre des fonctionnalités avancées de gestion de flotte (mise à jour des firmware à distance) pour le déploiement.

Modification technique requise : L'ESP32 devrait être configuré pour communiquer directement avec l'Endpoint (point d'accès) du service Cloud via MQTT (protocole plus léger que HTTP) au lieu d'envoyer la requête à un script PHP. Le service Cloud se charge ensuite de stocker la donnée.

Solution Serveur Cloud (Alternative) : Comparatif des Coûts

Solution Cloud (Exemples)	Investissement Initial	Coût de Fonctionnement Annuel Estimé
AWS IoT Core	**0 €** (Pas de matériel local requis)	**~50 - 100$ €/an**
Azure IoT Hub	**0 €** (Pas de matériel local requis)	**~60 - 120$ €/an**

Comparatif composants

Cartes Principales (ESP32 avec Écran)

Critère	LILYGO T-Display ESP32	LILYGO T5 E-Paper V2.3.1	ESP32 DevKitC
Prix	~14€	~22–28€	~7–10€
Microprocesseur	ESP32 Dual-Core 240 MHz	ESP32 Dual-Core 240 MHz	ESP32 Dual-Core 240 MHz
Mémoire flash	4 Mo QSPI	4 Mo	4 à 16 Mo selon version
RAM	520 kB	520 kB	520 kB
Affichage	IPS TFT 1.14″ ST7789	E-Paper 2.13″ (SSD1680)	Aucun
Interface I²C	Oui	Oui	Oui
Autres interfaces	UART, SPI, PWM, ADC, I²S	SPI, I²C	UART, SPI, I²C, ADC
WiFi	802.11 b/g/n	802.11 b/g/n	802.11 b/g/n
Bluetooth	BT 4.2 + BLE	BT 4.2 + BLE	BT 4.2 + BLE
Consommation active	> 60 mA	26.4 mW lors refresh (ultra faible au repos)	~50–70 mA
Consommation sommeil	~120 µA	Très basse (E-Paper)	10–150 µA selon mode
Tension fonctionnement	2.7–4.2V	3.3V	5V USB ou 3.3V régulé
Point fort principal	Écran couleur réactif	Ultra faible conso / lisible soleil	Très documentée, stable
Point faible principal	Conso élevée	Refresh lent (8s)	Pas d'écran, nécessite module externe
Pertinence SAE	Solution retenue	Non retenue (rafraîchissement trop lent)	Bonne alternative "budget"

Comparatif composants

Capteurs de Dioxyde de Carbone (CO2)

Produit	Prix Estimé (€)	Mesure	Plage Mesure CO2	Précision Typique CO2	Interfaces Clés
Sensirion SCD30	**27**	T/H/CO2	400–10000 ppm	±(30 ppm+3% MV)	I²C, UART 3
MH-Z19B (Module)	20	CO2	0–5000 ppm	±50 ppm+5% MV	UART, PWM 4
SCD40	18	T/H/CO2	400–2000 ppm	±(50 ppm + 5%)	I²C
SCD41	20	T/H/CO2	400 – 5000 ppm	±(40 ppm ±5.0 %)	I²C

Comparatif composants

Capteurs Température (T) et Humidité (H)

Produit	Prix Estimé	Mesures	Plage Temp.	Précision Temp.	Plage Hum.	Précision Humidité	Interfaces Clés
Capteur DHT22 AM2302 + PCB	**~6€**	T/H	-40 °C à 80 °C	~0,5°C	0 à 100%	~2%	1-Wire (Bus Digital)
DFRobot I2C CHT8305C Temp/ Hum Sensor	**~7,12 €**	T/H	-20 °C à 85 °C	~0,3°C	0 à 100 %	~2%	I²C
Adafruit SHT45 Precision Temp Sensor	**12,96€**	T/H	−40-125 °C	±0.1°C	0% à 100 %	±1.0%	I²C
Pro - Grove	**12,00€**	T/H	-40°C à 80°C	± 0.3°C	20% à 90%	~2%	I²C
Sensirion SCD40	**18€**	T/H/CO2	-10°C à 60°C	± 0.8°C	0% à 95%	~6%	I²C
Sensirion SCD41	**20€**	T/H/CO2	-10°C à 60°C	± 0.8°C	0% à 95%	~6%	I²C
Sensirion SCD30	**27€**	T/H/CO2	- 40°C – 70°C	± 0.4°C	0% - 100%	±3%	I²C

Comparatif composants

Configuration "Performance/Facilité"

Cette solution est le choix de la simplicité et de la fiabilité industrielle. Elle repose sur le capteur Sensirion SCD30, une marque reconnue mondialement.

- **L'installation est grandement simplifiée, car le SCD30 gère l'acquisition du CO2, de la Température et de l'Humidité via une seule interface I^2C. Sensirion offre une documentation complète et des bibliothèques Arduino bien maintenues, assurant un développement logiciel plus rapide.**

Composant	Option choisie	Prix unitaire estimé
Carte Principale	LILYGO T-Display ESP32	**13,90€**
Capteur CO_2/T/H	Sensirion SCD30 Haut de gamme	**27€**
Divers	Câble, boitier	**5€**
COÛT TOTAL ESTIMÉ		**45,90€**

Cette option est recommandée pour un déploiement à grande échelle où la rapidité d'installation et la maintenance simplifiée priment. Le coût d'achat est amorti par un coût de développement et de maintenance inférieur

Comparatif composants

Configuration "Compromis/Budget" (Multi-capteurs)

Cette solution représente le choix de la spécialisation, devenant la configuration la plus chère (46,86€).

- **Point Fort : Précision en T/H. L'utilisation du Adafruit SHT45 (basé sur le Sensirion SHT4x) garantit une précision Température et Humidité bien plus performante (souvent ±0,1◦C et ±1% RH) que le SCD30. Ce découplage permet d'obtenir des données T/H de qualité pour l'analyse de l'atmosphère ambiante.**
- **Point Critique : CO2 Moins Bon et Montage Conséquent.**
  a. **Précision CO2 : La précision du CO2 est limitée à celle du MH-Z19B (environ ±50 ppm), ce qui est moins précis que le SCD30 officiel.**
  b. **Montage/Documentation : Le montage est le plus conséquent : il faut gérer deux capteurs distincts sur des protocoles différents (SHT45 en I^2C, MH-Z19B potentiellement en UART ou PWM). Bien que la documentation Adafruit soit excellente, l'intégration de deux modules complique le câblage et le logiciel, nécessitant un temps de développement accru.**

Composant	Option choisie	Prix unitaire estimé
Carte Principale	LILYGO T-Display ESP32	**13,90**
Capteur CO_2	MH-Z19B	**20€**
Capteur Température/ Humidité	Adafruit SHT45 Precision Temp Sensor	**12,96€**
Divers	Câble, boitier	**5€**
COÛT TOTAL ESTIMÉ		**51,86€**

Configuration "Compromis/Budget" (Multi-capteurs)

Cette option vise à obtenir un capteur pour le température et pour l'humidité plus performant en sacrifiant de la précision par rapport à la mesure du CO2

Analyse des Solutions d'Alimentation

Comparatif d'Alimentation : Filaire vs. Batterie

Le critère de mobilité est le facteur déterminant pour l'alimentation.

Critère	Alimentation Filaire (Secteur 5V)	Alimentation Batterie (Li-Po/Li-ion)
Mobilité	Nulle (Fixe)	Totale (Déplacement libre du SA)
Autonomie	Illimitée	Limitée voir très limité.
Fiabilité	Très haute (Source constante)	Dépend de la batterie.
Complexité	Très faible	Moyenne (gestion charge, protection)
Sécurité	Bonne	Risques batterie (surcharge, usure)

Choix Préféré : Alimentation filaire
Le choix filaire nous parait plus judicieux. Comme notre ESP32 fonctionnera en continu et qu'on ne pourra pas utiliser le mode Deep Dleep car on veut actualiser nos données en local fréquemment, l'autonomie risque d'être grandement impacté..

Critères utilisés pour choisir la tension d'alimentation

Pour choisir la tension d'alimentation adaptée à notre Système d'Acquisition, nous retenons uniquement les critères techniques indispensables : compatibilité, stabilité du courant, simplicité d'intégration et fiabilité.

La carte TTGO T-Display ESP32 fonctionne en interne en 3,3 V mais est conçue pour être alimentée en 5 V via USB, grâce à son régulateur intégré. Une alimentation directe en 3,3 V ou supérieure à 5 V nécessiterait l'ajout de régulateurs externes, augmentant les risques d'instabilité ou de mauvais dimensionnement.

Le système doit supporter les pics de consommation du Wi-Fi, de l'affichage et du capteur NDIR SCD30. La consommation totale du SA est estimée à :

- **ESP32 actif : ~70 à 80 mA**
- **Écran TFT : ~20 à 40 mA**
- **SCD30 : ~19 mA**
- **Pics Wi-Fi : jusqu'à 250–300 mA**

Ce qui donne une consommation continue d'environ : Total ≈ 110 à 140 mA
avec des pics transitoires pouvant atteindre : ≈ 250–300 mA

L'alimentation choisie doit donc rester stable au-delà de ces valeurs. Une alimentation filaire 5 V / 1 A offre une marge largement suffisante, tout en étant standard, simple à intégrer et compatible nativement avec la TTGO.

Cette approche garantit une alimentation fiable, sans conversion supplémentaire, et parfaitement adaptée au fonctionnement continu du SA.

Comparaison détaillée des tensions possibles (3,3 V / 5 V / 12 V)

Afin de sélectionner la tension d'alimentation la plus adaptée à notre Système d'Acquisition, nous comparons uniquement les options réellement envisageables : 3,3 V, 5 V et 12 V.

Critère	Alimentation 3,3 V	Alimentation 5 V (USB)	Alimentation 12 V et +
Compatibilité avec la TTGO T-Display	Non adaptée : alimentation externe 3,3 V nécessaire, risque d'instabilité.	Compatible nativement via USB ; régulation vers 3,3 V intégrée.	Non compatible : nécessite conversion 12 V → 5 V → 3,3 V.
Stabilité face aux pics de courant (Wi-Fi)	Forte sensibilité : risque de chute de tension et redémarrages.	Très stable : les alimentations 5 V / 1 A absorbent les pics sans problème.	Dépend entièrement de la qualité du convertisseur DC-DC.
Simplicité d'intégration	Demande un régulateur externe dédié et un câblage irréprochable.	Très simple : connexion directe via USB-C / micro-USB.	Complexe : ajout obligatoire d'un convertisseur et filtrage.
Disponibilité	Peu courant en alimentation secteur.	Très courant : chargeurs USB standards.	Surtout utilisé en milieu industriel.
Sécurité électrique	Correcte mais dépend de la stabilité de l'alim externe.	Haute : tension faible, norme USB, courant limité.	Nécessite protections additionnelles (fusible, filtrage).
Adaptation aux longueurs de câble	Mauvaise : chute de tension rapide.	Adaptée aux longueurs courtes à moyennes.	Bonne pour longues distances, mais conversion obligatoire.
Pertinence pour notre projet	Non retenue : complexité sans bénéfice.	Solution optimale : simple, fiable, prévue pour la TTGO.	Inutilement complexe pour un SA local.

Solution Finale Retenue : Configuration Technique

Le choix final se porte sur la Configuration "Performance/Facilité" (ESP32 + SCD30) combinée à la Communication Wi-Fi et à l'alimentation en filaire.

Axe	Composant / Solution Retenue	Justification du Choix
Carte Principale	**TTGO T-Display ESP32**	Meilleur rapport coût/ performance intégrant l'écran et le Wi-Fi.
Capteurs	**Sensirion SCD30**	Solution 3-en-1 sur I²C. Simplicité de montage, code réduit et meilleure précision CO2
Communication	**Wi-Fi 802.11**	Utilisation de l'infrastructure Eduroam et intégration dans la carte.
Alimentation	**Filaire**	Essentielle l'alimentation continue du SA.
Coût Matériel Final	**45,90 €** par unité	Coût initial le plus faible pour une solution de marque fiable.

Architecture Technique Finale - Solution Retenue

Configuration Complète du Système

SA (ESP32 + SCD30)
↓ Wi-Fi HTTPS/JSON
Raspberry Pi (Nginx + PHP)
↓ Requêtes SQL
Base de données MariaDB
↓ Interface web
Visualisation données temps réel

Paramètres de Fonctionnement — Synthèse

Acquisition & Transmission

- Acquisition locale : toutes les 30 s
- Envoi serveur : toutes les 6 min
- Format : JSON structuré
- Gestion erreurs : buffer local + reconnexion auto

Communication

- Réseau : Wi-Fi Eduroam (WPA2-Enterprise)
- Protocole : HTTPS + token par device

Backend

- API d'ingestion : PHP natif
- Interface web : Symfony
- Serveur : Nginx + PHP

Base de données

- MariaDB

Infrastructure

- Matériel : Raspberry Pi 4 (4 Go RAM)
- OS : Raspberry Pi OS 64 bits

Sources des composants

Mouser.fr

Semageek.com

Amazon.fr

adafruit.com

gotronic.fr

sensirion.com

Glossaire des Acronymes

Acronyme	Signification complète	Description
SA	Système d'Acquisition	L'unité complète de mesure (Capteurs + ESP32).
I²C	Inter-Integrated Circuit	Bus de communication série à 2 fils (SDA, SCL) utilisé par les capteurs numériques.
UART	Universal Asynchronous Receiver-Transmitter	Protocole de communication série asynchrone (2 fils : RX, TX).
NDIR	Non-Dispersive Infrared	Technologie optique utilisée par le capteur SCD30 pour mesurer le CO_2.
ASC	Automatic Self-Calibration	Mécanisme interne du SCD30 pour auto-corriger la dérive en utilisant un point de référence (air frais, 400 ppm).
OTA	Over-The-Air	Protocole de mise à jour du firmware à distance via Wi-Fi.
Gateway	Passerelle	Dispositif intermédiaire utilisé pour traduire le protocole BLE en protocole Internet (HTTP).
GATT	Generic Attribute Profile	Structure de communication utilisée par le protocole Bluetooth Low Energy (BLE).
LEMP	Linux, Engine X (Nginx), MariaDB, PHP	Pile logicielle utilisée pour l'hébergement du serveur web sur le Raspberry Pi.
HMAC	Hash-based Message Authentication Code	Signature cryptographique utilisée pour garantir l'intégrité des données transmises.