

Sommaire

Introduction	2
Décomposition de cahier des charges	3
1 Introduction	3
1.1 Modèle SART	4
Réalisation	6
2 Hardware	6
2.1 Matériels utilisés	6
2.2 Montage	8
3 Software	10
3.1 Structure mesure	10
3.2 Principe du modèle producteurs multiples - consommateur unique	10
3.3 Tâches productrices	11
3.4 Tâches consommatrices	11
3.5 Mesure de distance avec le capteur HC-SR04 (framework ESP-IDF)	12
Test et résultats	13
4 Mode d'emploi	13
4.1 Analyse et discussion	13

Introduction

Dans le cadre du module SETR 2A, nous avons réalisé un projet visant à concevoir et mettre en œuvre un système embarqué de mesure et d’affichage des niveaux de dioxyde de carbone, de température et d’hygrométrie, activé par la détection de présence. Ce projet, basé sur l’utilisation de deux microcontrôleurs ESP32 et de divers capteurs, a permis d’explorer les concepts fondamentaux des systèmes embarqués, tels que la communication inter-puces (UART), la synchronisation des tâches, et la gestion des données en temps réel. En adoptant une approche modulaire et en respectant le modèle producteurs-consommateurs, ce système illustre une solution pratique pour la surveillance environnementale dans des espaces intérieurs. Ce rapport détaille les étapes de conception, de développement et de test du projet, tout en présentant les choix techniques effectués et les résultats obtenus.

Décomposition de cahier des charges

1 Introduction

Le projet vise à réaliser un système embarqué mesurant et affichant le CO₂, la température et l'hygrométrie, activé par détection de présence. Voici les exigences principales reformulées :

Mesure des paramètres

Un ESP32 OLED, sous framework Arduino, mesure température et humidité. Un second ESP32, sous ESP-IDF, gère le CO₂ et la détection par ultrasons (HC-SR04), assurant une collecte modulaire des données.

Communication

Les deux ESP32 communiquent via UART pour transmettre les données du module de mesure vers l'affichage.

Détection de présence

Le capteur HC-SR04 déclenche le système en cas de présence, optimisant l'usage énergétique.

Architecture logicielle

Le modèle producteurs-consommateurs est implémenté : les capteurs (CO₂, température, humidité) produisent des données, consommées par l'écran OLED. Exclusion mutuelle et synchronisation garantissent la cohérence.

Fonctionnalité supplémentaire

Un serveur web est ajouté pour afficher les données à distance.

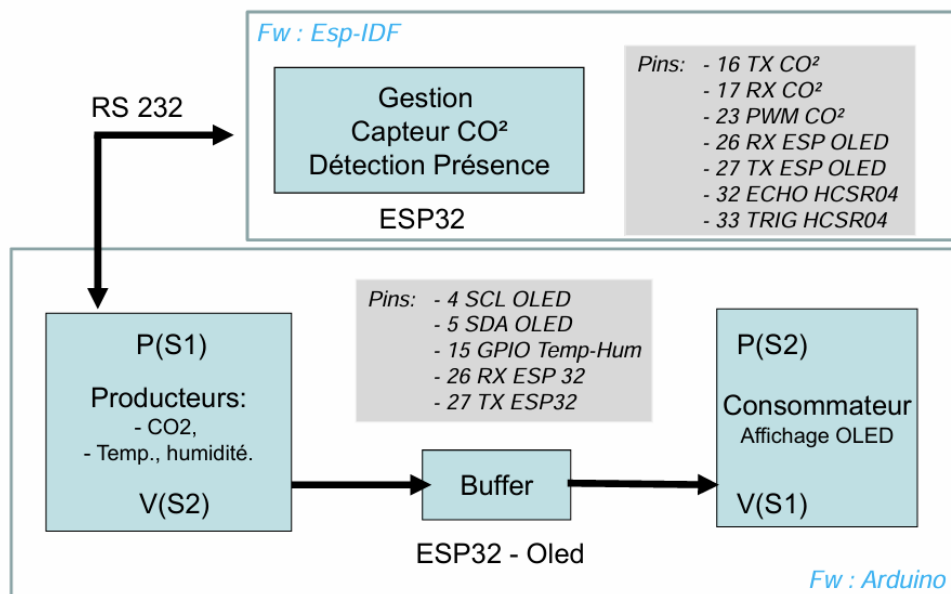


FIGURE 1 – Architecture simplifié du système

1.1 Modèle SART

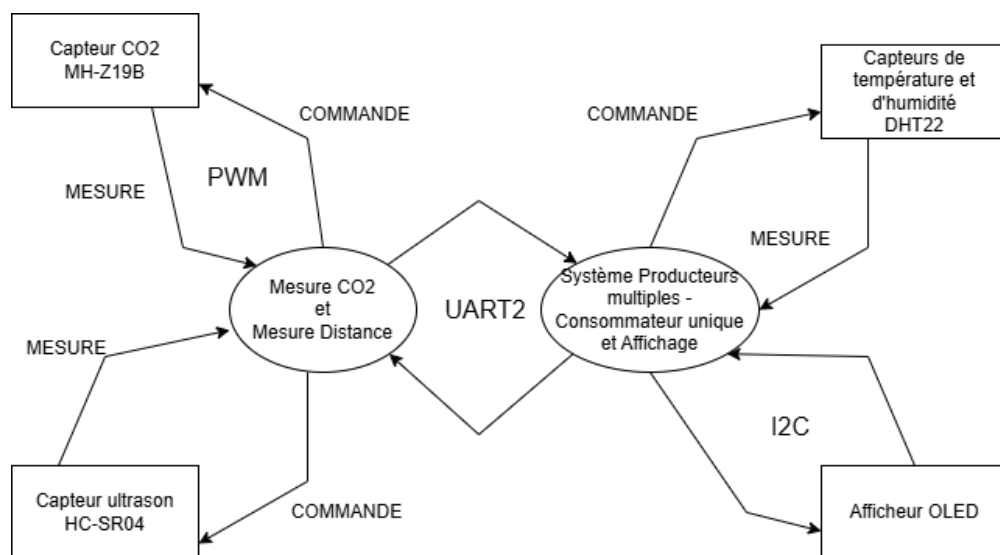


FIGURE 2 – Diagramme du contexte

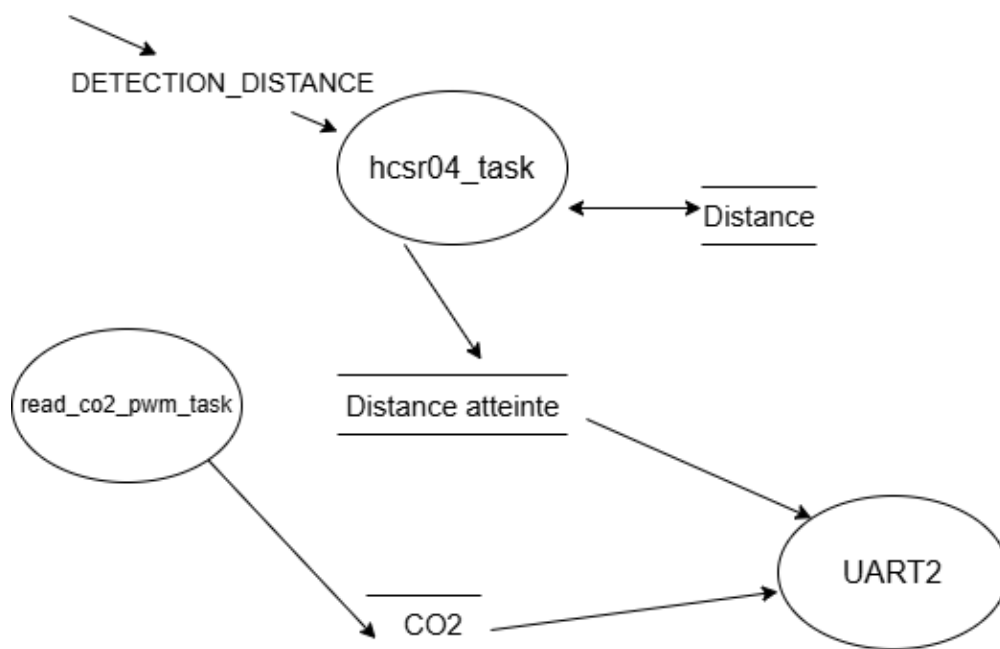


FIGURE 3 – DFD0 et DFC0 : Mesure distance et CO2 par la carte ESP32

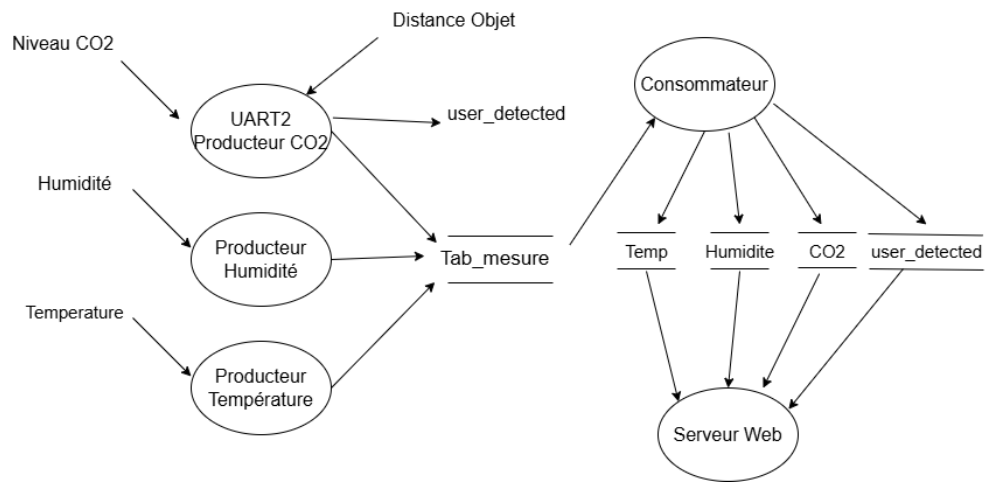


FIGURE 4 – DFD1 et DFC1 : Système producteurs multiples - consommateur unique

Réalisation

2 Hardware

2.1 Matériels utilisés

ESP 32 :

on a utilisé microcontrôleur ESP32 est équipé d'une fréquence de processeur allant jusqu'à 240 MHz et dispose d'une mémoire flash de 16 MB. Il prend en charge plusieurs interfaces de communication, telles que UART, SPI, I2C, GPIO, Wi-Fi et Bluetooth, offrant une grande flexibilité pour les applications embarquées.

On a aussi utilisé un autre microcontrôleur ESP32 équipé d'un écran OLED d'une résolution de 128 x 64 pixels, permettant un affichage direct des données mesurées.



FIGURE 5 – ESP32 DEV-KIT



FIGURE 6 – ESP32 WROOM

Capteur CO₂ (MH-Z19B)

Le capteur MH-Z19B est dédié à la mesure du dioxyde de carbone, avec une plage de mesure allant de 0 à 5 000 ppm. Il peut être interfacé via UART ou une mesure de PWM, offrant une solution précise pour la surveillance de la qualité de l'air.

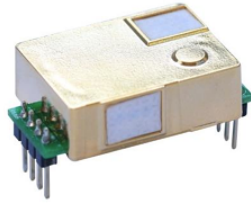


FIGURE 7 – MH-Z19B

Capteur à ultrasons HC-SR04

Le capteur HC-SR04 permet de mesurer des distances dans une plage de 2 cm à 400 cm. Il fonctionne sous une tension de 5 V et utilise une interface GPIO, ce qui le rend adapté pour la détection de présence dans le cadre du projet.



FIGURE 8 – HC-SR04

Capteur DHT22

Le capteur DHT22 est utilisé pour mesurer la température et l'humidité. Il est alimenté sous une tension de 3,3 à 6 Vcc. Sa plage de mesure s'étend de -40 à +80 °C pour la température et de 0 à 100 % pour l'humidité, avec une connexion via une interface GPIO.

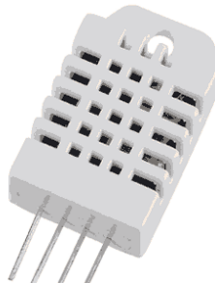


FIGURE 9 – DHT22

2.2 Montage

Câblage du premier ESP32 avec le capteur ultrason et le capteur CO2 (ESP-IDF)

- **Capteur à ultrasons HC-SR04 :**
 - TRIGGER (sortie) : GPIO 33.
 - ECHO (entrée) : GPIO 32.
 - Alimentation : VCC à 3.3 V, GND à la masse de l'ESP32.
- **Capteur CO₂ MH-Z19B (via PWM) :**
 - PWM (entrée) : GPIO 4.
 - Alimentation : VCC à 5 V, GND à la masse de l'ESP32.
- **Communication UART avec le second ESP32 :**
 - TXD : GPIO 17 (vers RX du second ESP32).
 - RXD : GPIO 16 (depuis TX du second ESP32).

Câblage du second ESP32-OLED avec le capteur DHT22 et l'écran OLED (Arduino Framework)

- **Capteur DHT22 :**
 - Données : GPIO 18.
 - Alimentation : VCC à 3,3 V , GND à la masse de l'ESP32-OLED.
 - Résistance de pull-down : 1 k Ω entre la masse et la broche de données.
- **Écran OLED (via I2C) :**
 - SDA : GPIO 5.
 - SCL : GPIO 4.
- **Communication UART avec le premier ESP32 :**
 - TX : GPIO 17 (vers RX du premier ESP32).
 - RX : GPIO 16 (depuis TX du premier ESP32).

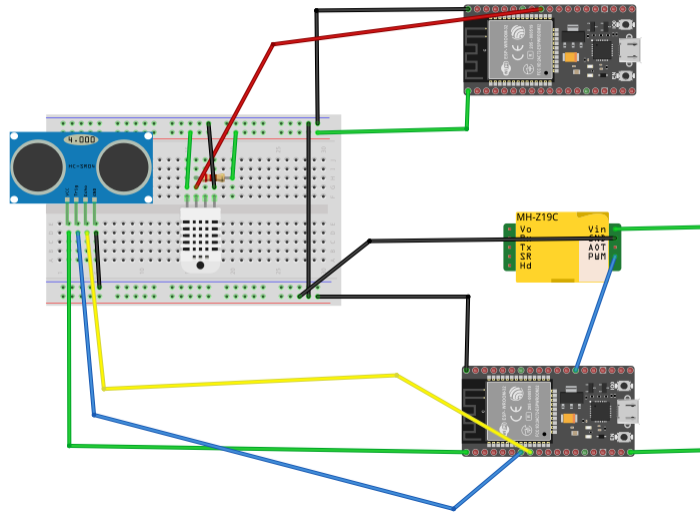


FIGURE 10 – Schéma simplifié de câblage

Communication entre les deux ESP32

- Connexion UART croisée :
 - ESP32 (TX, GPIO 17) → ESP32-OLED (RX, GPIO 16).
 - ESP32 (RX, GPIO 16) → ESP32-OLED (TX, GPIO 17).
- Masse commune : Les GND des deux ESP32 doivent être connectés ensemble.

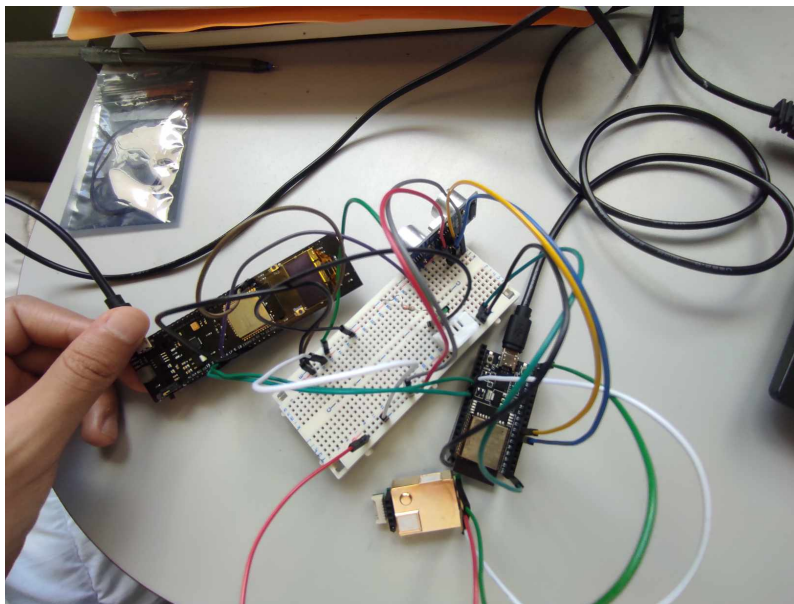


FIGURE 11 – photo du système

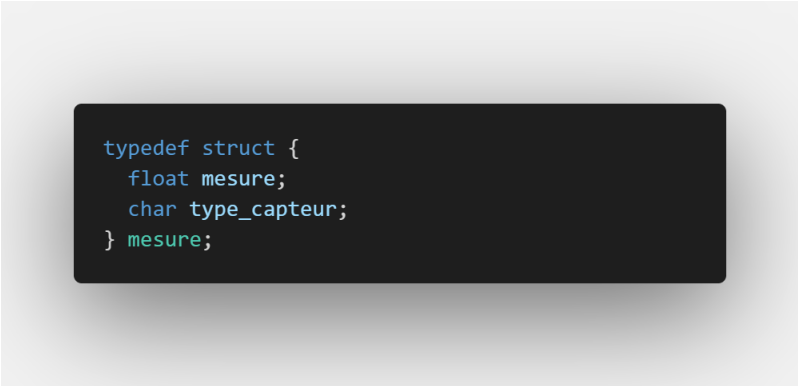
3 Software

3.1 Structure mesure

La structure `mesure` est définie pour standardiser le stockage des données provenant des différents capteurs. Elle est composée de deux champs :

- `mesure` : un `float` représentant la valeur mesurée (température, humidité ou CO₂).
- `type_capteur` : un `char` indiquant le type de capteur ('T' pour température, 'H' pour humidité, 'C' pour CO₂).

Cette structure est utilisée dans un tableau circulaire `tab_mesure` de taille `TAILLE_MAX` (30 éléments), permettant une gestion efficace des données selon le modèle producteurs-consommateurs.



```
typedef struct {  
    float mesure;  
    char type_capteur;  
} mesure;
```

FIGURE 12 – Structure mesure

3.2 Principe du modèle producteurs multiples - consommateur unique

Le code implémente le modèle **producteurs multiples - consommateur unique** pour gérer les données des capteurs (température, humidité, CO₂). Trois tâches productrices (`vProducteurTemperature`, `vProducteurHumidite`, `vUARTReceiver`) génèrent des données et les placent dans le tableau circulaire `tab_mesure`. Une unique tâche consommatrice (`vConsommateur`) extrait ces données pour les traiter. Ce modèle est soutenu par des mécanismes de synchronisation :

- **Sémaphores de comptage** : `s1` (initialisé à `TAILLE_MAX`) contrôle les emplacements libres dans le tableau, et `s2` (initialisé à 0) indique les données disponibles pour la consommation.
- **Exclusion mutuelle** : Le sémaphore `mutex` protège l'accès concurrent au tableau `tab_mesure`, évitant les corruptions de données lors des écritures et lectures simultanées.
- **Gestion circulaire** : L'indice `table_pointer` est incrémenté modulo `TAILLE_MAX` pour gérer le tableau comme une file circulaire, optimisant l'utilisation de la mémoire.

Ce modèle garantit une communication asynchrone et efficace entre les producteurs et le consommateur, tout en évitant les blocages grâce à une synchronisation rigoureuse.

3.3 Tâches productrices

Les tâches productrices collectent les données des capteurs et les placent dans le tableau `tab_mesure`, en respectant le modèle producteurs-consommateurs avec des sémaphores pour la synchronisation.

Tâche `vProducteurTemperature`

Cette tâche, exécutée sur le second ESP32-OLED, lit la température à partir du capteur DHT22 toutes les 250 ms. Elle utilise la bibliothèque `DHTesp` pour récupérer la température via la méthode `getTempAndHumidity()`. La valeur est encapsulée dans une structure `mesure` avec le type 'T', puis insérée dans `tab_mesure` après avoir pris les sémaphores `s1` (pour vérifier l'espace disponible) et `mutex` (pour éviter les accès concurrents). Une fois la donnée ajoutée, elle libère `mutex` et `s2` pour signaler aux consommateurs qu'une nouvelle donnée est disponible.

Tâche `vProducteurHumidite`

Similaire à `vProducteurTemperature`, cette tâche lit l'humidité du capteur DHT22 toutes les 250 ms. La valeur est stockée dans une structure `mesure` avec le type 'H' et insérée dans `tab_mesure` en suivant le même mécanisme de synchronisation avec les sémaphores `s1`, `mutex`, et `s2`.

Tâche `vUARTReceiver (CO2 et détection de présence)`

Exécutée sur le second ESP32-OLED, cette tâche reçoit les données du premier ESP32 via UART (`SerialSensor`). Elle gère deux types de messages :

- Détection de présence : si le message reçu est « Object detected at 50 cm ! », elle met à jour la variable `user_detected` et active l'affichage.
- Mesure de CO₂ : si le message est une valeur numérique, elle est convertie en `float`, encapsulée dans une structure `mesure` avec le type 'C', et insérée dans `tab_mesure` avec synchronisation via `s1`, `mutex`, et `s2`.

La tâche gère également l'état de l'écran OLED, le désactivant après un délai de 3 secondes (`proximityTimeout`) si aucune présence n'est détectée.

3.4 Tâches consommatrices

Les tâches consommatrices extraient les données du tableau `tab_mesure` pour les utiliser dans l'affichage ou la transmission.

Tâche `vConsomateur`

Cette tâche, exécutée sur le second ESP32-OLED, consomme les données de `tab_mesure` toutes les 250 ms. Elle prend le sémaphore `s2` pour vérifier la disponibilité des données, extrait la valeur courante, et libère `s1` pour signaler un emplacement libre. Selon le `type_capteur`, elle met à jour les variables globales `temp`, `humidite`, ou `co2`, qui sont ensuite utilisées pour l'affichage.

Tâche `vSendWebSocketData`

Exécutée sur le second ESP32-OLED, cette tâche envoie les données (`temp`, `humidite`, `co2`, `user_detected`) via WebSocket toutes les 1 000 ms. Elle utilise `ArduinoJson` pour formater les données en JSON, puis les transmet à tous les clients connectés via `ws.textAll()`. Cela permet une visualisation en temps réel sur une interface web.

3.5 Mesure de distance avec le capteur HC-SR04 (framework ESP-IDF)

Le premier ESP32 utilise le **framework ESP-IDF** pour gérer la mesure de distance via le capteur à ultrasons HC-SR04, dans la tâche `hcsr04_task`. Voici les étapes principales :

Configuration matérielle et interruption

Dans `app_main()`, les broches GPIO 33 (`TRIGGER_PIN`) et GPIO 32 (`ECHO_PIN`) sont configurées respectivement en sortie et en entrée. Une interruption (`GPIO_INTR_ANYEDGE`) est attachée à la broche `ECHO_PIN` via `gpio_isr_handler_add()`. L'ISR `echo_isr_handler()` enregistre les temps de début et de fin du signal Echo (en microsecondes) à l'aide de `esp_timer_get_time()`.

Déclenchement et mesure

La fonction `trigger_sensor()` génère une impulsion de 10 μs sur `TRIGGER_PIN` pour déclencher le capteur. La tâche `hcsr04_task` calcule ensuite la durée de l'écho (`end_time - start_time`) et détermine la distance en cm avec la formule : $\text{distance} = (\text{durée} \times \text{vitesse du son}) / 2$, où la vitesse du son est définie à 0,0343 cm/ μs (`SOUND_SPEED`).

Détection de présence

Si la distance mesurée est inférieure à 50 cm (`DETECTION_DISTANCE`), un message « Object detected at 50 cm ! » est envoyé via UART (`UART_NUM_2`, broches TX : GPIO 17, RX : GPIO 16) au second ESP32. L'envoi est protégé par un sémaphore `uart_mutex` pour éviter les conflits d'accès. La tâche s'exécute toutes les 500 ms, assurant une détection régulière de présence.

Test et résultats

4 Mode d'emploi

Pour mettre en œuvre le système, suivez les étapes ci-dessous dans l'ordre indiqué :

1. Vérifiez que le câblage est correct, en prêtant une attention particulière aux masses communes.
2. Flashez le programme dédié à l'ESP32 DevKit sur cette carte.
3. Coupez l'alimentation de l'ESP32 DevKit après le flashage.
4. Flashez le programme dédié à l'ESP32-WROOM sur cette carte.
5. Configurez le système de fichiers SPIFFS sur l'ESP32-WROOM pour la page `index.html`, en exécutant les commandes suivantes dans PlatformIO :

```
pio run --target buildfs  
pio run --target uploadfs
```

La première commande construit le système de fichiers SPIFFS, et la seconde le téléverse sur la carte.

6. Alimentez les deux cartes ESP32.
7. Appuyez sur le bouton EN de l'ESP32 pour redémarrer la carte.
8. Attendez que l'ESP32-WROOM se connecte au réseau Wi-Fi (vérifiez les identifiants réseau dans le code).
9. Une fois la connexion Wi-Fi établie, le système est prêt à fonctionner.

4.1 Analyse et discussion

Les tests ont démontré que le système répond aux exigences principales du cahier des charges. La détection de présence est efficace dans la plupart des scénarios, bien que des améliorations pourraient être envisagées pour gérer les surfaces réfléchissantes, par exemple en ajustant la sensibilité du capteur HC-SR04. Les mesures des capteurs sont précises et fiables pour une utilisation dans un environnement intérieur, mais des calibrations supplémentaires pourraient réduire les écarts observés. Le serveur web offre une interface pratique pour la visualisation des données, bien que la stabilité de la connexion Wi-Fi soit un facteur critique à surveiller.

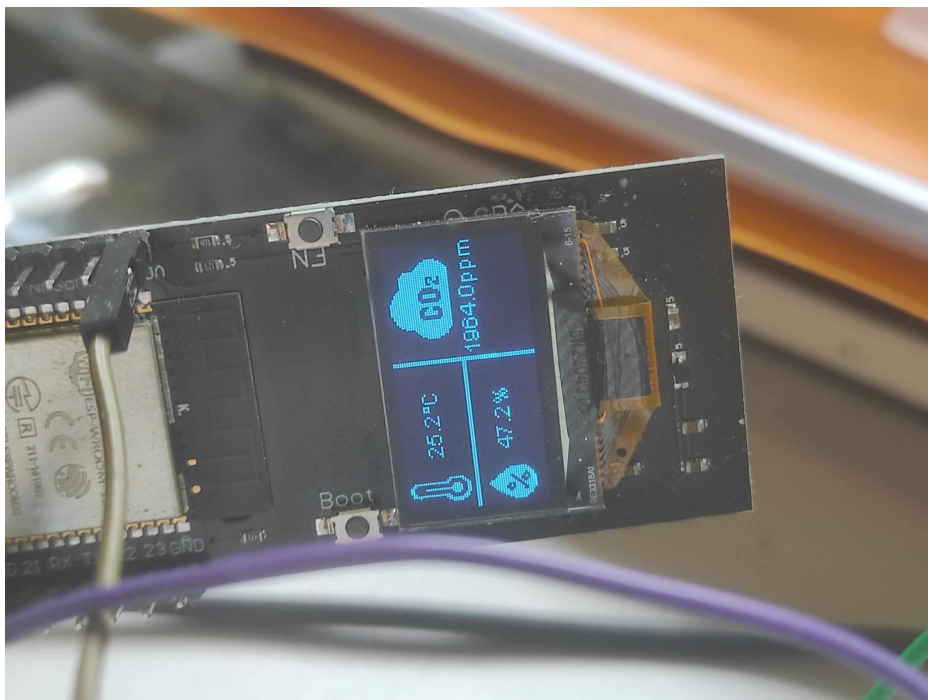


FIGURE 13 – Affichage sur l'écran OLED

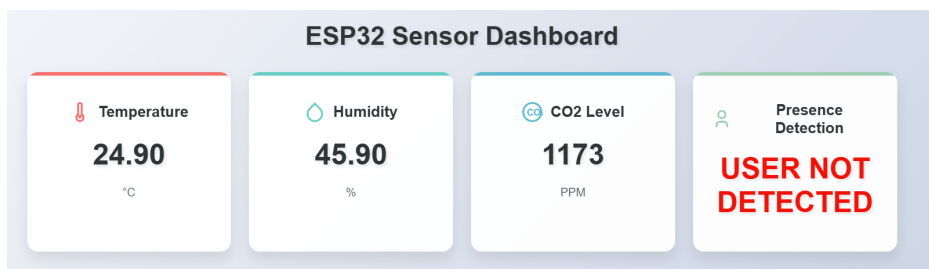


FIGURE 14 – Page web

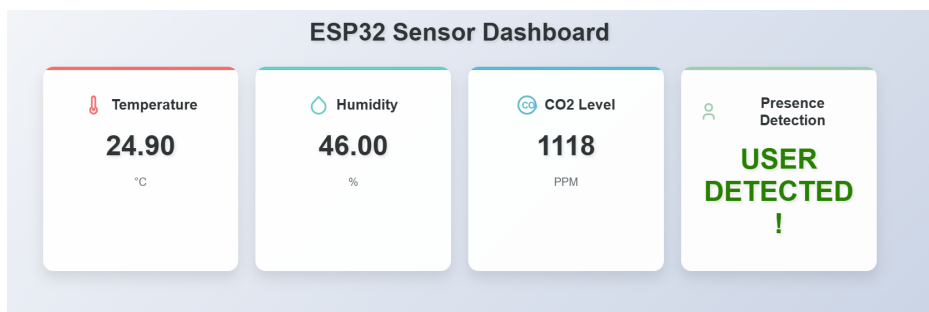


FIGURE 15 – Page web

Index des figures

1	Architecture simplifié du système	3
2	Diagramme du contexte	4
3	DFD0 et DFC0 : Mesure distance et CO2 par la carte ESP32	4
4	DFD1 et DFC1 : Système producteurs multiples - consommateur unique . .	5
5	ESP32 DEV-KIT	6
6	ESP32 WROOM	6
7	MH-Z19B	7
8	HC-SR04	7
9	DHT22	7
10	Schéma simplifié de câblage	9
11	photo du système	9
12	Structure mesure	10
13	Affichage sur l'écran OLED	14
14	Page web	14
15	Page web	14