جامعة ابن طفيل
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵙⵉ ⵄⵣⵉⵣⴰⵏ
Ibn Tofaïl University

المدرسة العليا للتكنولوجيا
ⵜⵉⵣⵉⵔⴰⵏ ⵜⴰⵍⴰⵣⵉⵏⴰⵜ ⵏ ⵜⵣⵔⵉⵎⵉⵜ
Higher School of Technology

Filière : Informatique Industrielle et Robotique **(IIR)**

Module : IOT

Conception et Réalisation d'une Sonde IoT pour la Surveillance de la Qualité de l'Air Intérieur

Réalisées par :
AKIM Fatima
BOUSSAQ Fatima
AIT SALAH Noura

Encadré par :
Prof MAZOUIARI Abde alkadder

Année Universitaire : 2025-2026

Sommaire

LISTE DE FIGURES

Introduction

Aujourd'hui, nous passons la majeure partie de notre temps à l'intérieur : chez nous, au travail, à l'école. Mais savons-nous vraiment si l'air que nous y respirons est bon pour notre santé ?

En effet, l'air intérieur peut parfois être plus pollué que l'air extérieur. Des gaz comme le CO_2, des particules ou des produits chimiques invisibles peuvent s'accumuler dans nos pièces fermées. Cela peut causer des maux de tête, de la fatigue, des allergies ou des problèmes de concentration.

Pourtant, il est difficile de savoir si l'air est de bonne qualité sans outil de mesure. Les systèmes professionnels sont souvent chers et compliqués à installer.

C'est pour cela que nous avons créé ce projet : une sonde intelligente, simple et peu coûteuse, qui surveille en permanence la qualité de l'air. Elle mesure le taux de CO_2, les polluants chimiques, la température et l'humidité. Les données sont affichées en temps réel sur un écran et envoyées sur une application smartphone, pour que chacun puisse agir et mieux aérer son intérieur.

L'objectif est de rendre la surveillance de l'air accessible à tous, pour protéger notre santé et améliorer notre bien-être au quotidien.

CHAPITRE I

Contexte Général du Projet

1. Introduction

Ce chapitre présente le cadre général du projet, en abordant l'importance de la qualité de l'air intérieur et les fondements technologiques de l'Internet des Objets (IoT). Il expose également la problématique et les principes de la solution proposée.

1.1 Contexte de la Qualité de l'Air Intérieur

La qualité de l'air intérieur est une problématique cruciale, car nous passons environ 90% de notre temps dans des espaces fermés, qu'il s'agisse de foyers, de bureaux, d'écoles d'hôpitaux. Contrairement à l'air extérieur, souvent mieux ventilé, l'air intérieur peut contenir des concentrations plus élevées de polluants provenant de sources variées, comme les produits chimiques, les matériaux de construction, les équipements de chauffage ou encore les activités humaines.

• Importance de la Surveillance dans les Espaces Intérieurs :

La surveillance active de l'air intérieur repose sur des outils technologiques modernes capables de :

• Détecter les polluants invisibles, tels que le monoxyde de carbone (CO) et les COV, avant qu'ils n'atteignent des seuils critiques.

• Fournir des alertes en temps réel, pour permettre une réaction rapide en cas de dépasse-ment des limites de sécurité.

• Optimiser les systèmes de ventilation, en ajustant automatiquement leur fonctionnement pour garantir une qualité d'air optimale tout en réduisant la consommation énergétique

• Prévenir les risques sanitaires , liés à une exposition prolongée aux polluants, améliorant ainsi le confort et la santé des occupants

1.2 IoT (Internet of Things)

1.2.1 Principe de la technologie Internet Des Objets (IoT)

L'Internet des Objets (IoT, pour "Internet of Things") désigne un réseau d'objets physiques interconnectés qui collectent et échangent des données via Internet. Ces objets peuvent être des capteurs, des dispositifs électroniques ou des machines, équipés de technologies embarquées comme des puces RFID, des processeurs et des logiciels.

- **Fonctionnement des Systèmes IoT :**

Le fonctionnement des systèmes IoT repose sur une chaîne d'étapes interdépendantes. Tout d'abord, la collecte des données est effectuée à l'aide de capteurs qui mesurent des paramètres spécifiques tels que la température, l'humidité ou la qualité de l'air. Ensuite, ces données sont transmises via des protocoles de communication comme Wi-Fi, Bluetooth ou LoRaWAN vers une plateforme centrale ou un cloud. Une fois réceptionnées, elles subissent un traitement et une analyse pour être stockées, visualisées et converties en informations exploitables à l'aide d'outils logiciels. Enfin, une prise de décision est effectuée, soit par les utilisateurs soit par des systèmes automatisés, permettant d'initier des actions comme l'envoi d'alertes ou l'ajustement de certains paramètres pour répondre aux besoins identifiés.

1.2.2 Architecture IoT :

L'architecture IoT se compose de plusieurs couches, chacune ayant un rôle spécifique dans son fonctionnement global :

1. Couche de perception : Elle inclut les capteurs et les actionneurs, chargés de collecter les données physiques (comme la température et l'humidité) et de les transmettre.

2. Couche de réseau : Elle assure la transmission des données vers une plateforme centrale via des protocoles tels que Wi-Fi, Zigbee ou LoRaWAN.

3. Couche de traitement : Cette couche, souvent représentée par le cloud ou des serveurs locaux, se charge du traitement, de l'analyse et du stockage des données.

4. Couche applicative : Il s'agit de l'interface utilisateur où les données sont présentées et où les actions automatisées (alertes, tableaux de bord, etc.) sont exécutées.

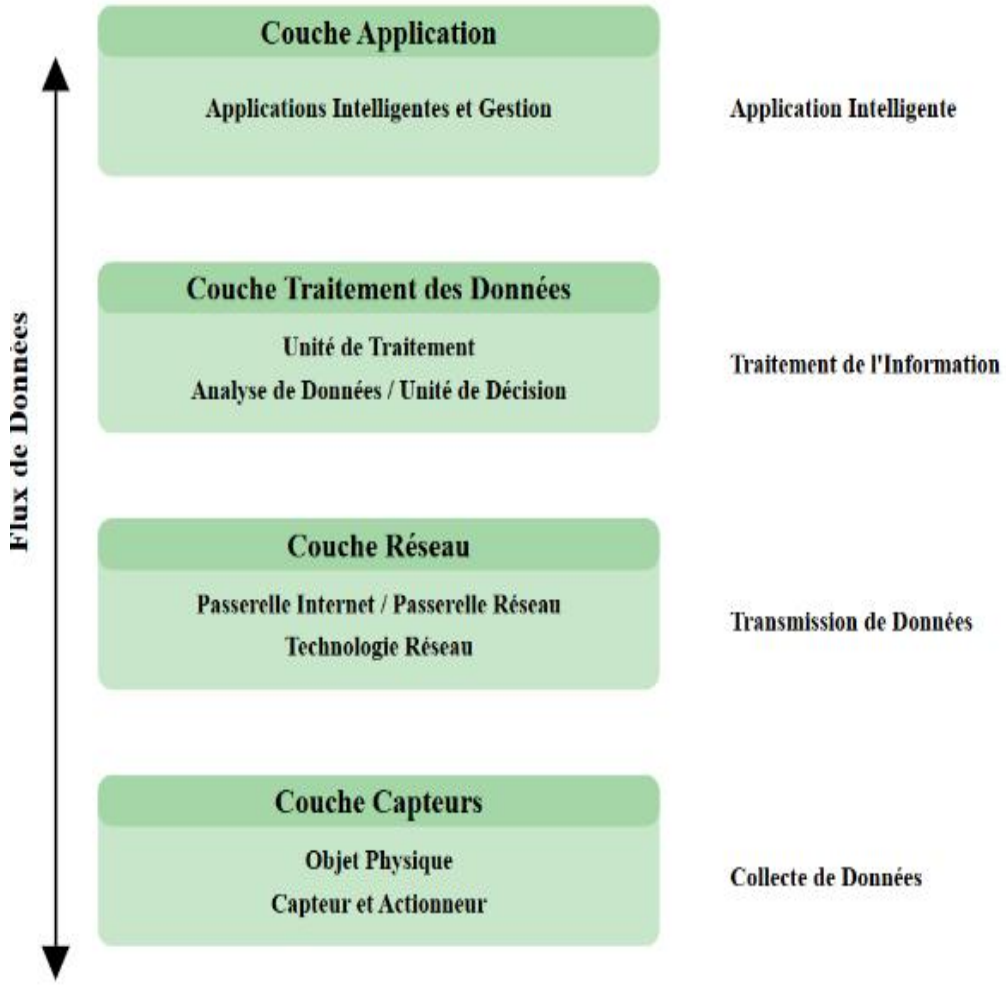

Figure 1: Architecture IoT

1.3 Présentation du Projet

1.3.1 Problématique

La qualité de l'air intérieur constitue aujourd'hui un enjeu majeur de santé publique, car les personnes passent une grande partie de leur temps dans des espaces fermés tels que les habitations, les salles de classe et les bureaux. L'exposition prolongée à un air pollué, riche en dioxyde de carbone (CO_2) et en gaz nocifs, peut provoquer des troubles respiratoires, une diminution de la concentration et divers problèmes de santé. Cependant, les systèmes de surveillance de la qualité de l'air restent souvent coûteux ou inexistants dans les environnements domestiques et éducatifs. Face à cette situation, ce projet vise à concevoir un système simple, fiable et économique basé sur un microcontrôleur ESP32, capable de mesurer et d'afficher en temps réel la qualité de l'air intérieur, afin de sensibiliser les utilisateurs et de contribuer à la prévention des risques liés à la pollution.

1.3.2 Principe du Projet

Ce projet repose sur la conception d'une sonde intelligente de surveillance de la qualité de l'air intérieur basée sur l'Internet des Objets. Le système utilise des capteurs pour mesurer en continu les paramètres environnementaux, notamment la concentration de CO_2 et de TVOC à l'aide du capteur MQ135, ainsi que la température et l'humidité grâce au capteur DHT. Ces données sont collectées et traitées par le microcontrôleur ESP32, qui joue le rôle de centre de commande du système. Grâce à la connexion Wi-Fi intégrée de l'ESP32, les données sont transmises en temps réel vers une plateforme en ligne pour le stockage, la visualisation et l'analyse. Lorsque les niveaux de pollution dépassent les seuils recommandés, le système génère des alertes afin de prévenir les occupants et favoriser une meilleure gestion de la qualité de l'air intérieur.

Figure 2 : l e principe du proget

Conclusion

Le Chapitre I a permis de poser les bases conceptuelles et techniques du projet. Il a mis en lumière l'urgence sanitaire liée à la pollution de l'air intérieur et justifié le recours à une solution IoT pour une surveillance continue, accessible et en temps réel. La problématique est clairement définie, ouvrant la voie à la conception d'un système intégré et intelligent.

CHAPITRE II

Matériaux et logiciels

1. Introduction

Ce chapitre détaille les composants matériels et les outils logiciels utilisés pour la réalisation du système. Il présente notamment le microcontrôleur ESP32, les capteurs MQ135 et DHT11, ainsi que les plateformes Blynk et Firebase.

1.1 Matériel utilisée

1.1.1 ESP32

L'ESP32 est un microcontrôleur programmable intégrant des fonctions de communication sans fil, notamment le Wi-Fi et le Bluetooth. Il est largement utilisé dans les projets d'Internet des Objets (IoT).

Dans ce projet, l'ESP32 constitue le cœur du système. Il assure la lecture des données provenant des capteurs, le traitement des informations, l'affichage des résultats sur l'écran LCD et l'envoi des données vers une plateforme en ligne.

Figure 3 : ESP32

Caractéristiques

- Tension de fonctionnement : 3.3 V
- Wi-Fi et Bluetooth intégrés
- Convertisseur analogique-numérique (ADC) 12 bits
- Processeur double cœur jusqu'à 240 MHz
- Faible consommation d'énergie

1.1.2 Capteur MQ135

Le capteur MQ135 est un capteur de gaz utilisé pour détecter la présence de gaz polluants dans l'air, notamment le dioxyde de carbone (CO_2) et les composés organiques volatils (TVOC).

Il permet de mesurer le niveau de pollution de l'air intérieur afin d'évaluer la qualité de l'air ambiant.

Figure 4 : Capteur MQ135

Caractéristiques

- Tension d'alimentation : 5 V
- Sortie analogique
- Détecte : CO_2, NH_3, benzène, alcool, fumée
- Temps de préchauffage : 24 à 48 heures
- Sensibilité moyenne, nécessite calibration

1.1.3 Capteur DHT11

Le capteur **DHT11** est un capteur numérique utilisé pour mesurer **la température** et **l'humidité relative de l'air**. Il est largement employé dans les projets électroniques et IoT en raison de sa simplicité et de son faible coût.

Dans ce projet, le capteur DHT11 sert à **mesurer la température et l'humidité de l'air intérieur**. Ces paramètres permettent de mieux interpréter la qualité de l'air et d'améliorer l'analyse des données fournies par le capteur de gaz.

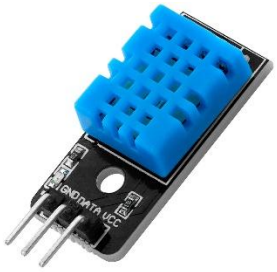

Figure 5 : Capteur DHT11

Caractéristiques techniques

- Type de capteur : Température et humidité
- Tension d'alimentation : 3.3 V à 5 V
- Plage de température : 0 à 50 °C
- Précision température : ±2 °C

- Plage d’humidité : 20 % à 80 %
- Précision humidité : ±5 %
- Type de sortie : Numérique (1 fil de données)
- Fréquence de mesure : 1 mesure par seconde
- Faible consommation d’énergie

1.1.4 Breadboard

La breadboard est une plaque de prototypage permettant de connecter des composants électroniques sans soudure.

Elle facilite le montage, les tests et les modifications du circuit électronique.

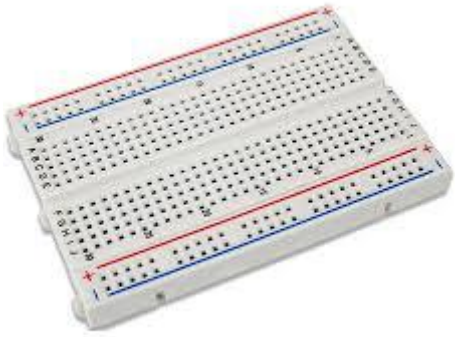

Figure 6 : Breadboard

Caractéristiques

- Réutilisable
- Connexion rapide
- Adaptée au prototypage

1.1.5 Fils de Connexion

Les fils de connexion (Dupont) servent à relier électriquement les différents composants du circuit.

Ils assurent la transmission de l’alimentation et des signaux entre les composants.

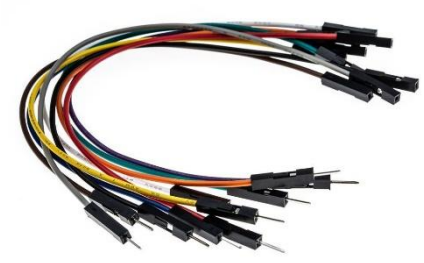

Figure 7 : Fils de Connexion

Caractéristiques

- Types : mâle-mâle, mâle-femelle
- Faible résistance
- Faciles à utiliser

1.1.6 Buzzer

Un buzzer est un composant électronique utilisé pour produire un signal sonore. Il permet de générer des bips ou des alertes sonores afin d'informer ou d'avertir l'utilisateur d'un événement particulier.

Dans les systèmes électroniques et les projets IoT, le buzzer est souvent utilisé comme dispositif d'alerte, par exemple pour signaler un danger, une erreur ou le dépassement d'un seuil prédéfini. Il peut être activé ou désactivé par un microcontrôleur comme l'ESP32 à l'aide d'un signal électrique.

Figure 8 : Buzzer

Caractéristiques

- Tension de fonctionnement : 3 V à 5 V
- Courant de fonctionnement : faible consommation
- Type : Buzzer actif ou buzzer passif
- Niveau sonore : environ 85 à 95 dB (selon le modèle)
- Fréquence de fonctionnement : 2 kHz à 4 kHz (buzzer actif)
- Commande : signal numérique depuis le microcontrôleur
- Temps de réponse : immédiat
- Utilisation : alertes et signalisations sonores
- Taille : compacte et légère
-

1.2 Les logiciels utilisés

Pour le développement et la mise en œuvre de notre projet IoT, plusieurs logiciels et plateformes ont été utilisés afin de programmer le microcontrôleur, collecter les données et les visualiser. Chacun de ces outils apporte des fonctionnalités spécifiques : l'IDE Arduino permet de coder et tester le microcontrôleur, Blynk facilite la création de l'application mobile pour le suivi en temps réel, et Firebase assure le stockage sécurisé

et la gestion des données. Cette section présente les principaux logiciels utilisés, leurs rôles dans le projet et leurs composants essentiels.

1.2.1 IDE (Arduino IDE)

L'IDE Arduino est un environnement de développement intégré qui permet de programmer facilement les microcontrôleurs tels que l'ESP32. Il offre un éditeur de code simple, un compilateur et un outil de téléversement du programme vers la carte. L'IDE intègre également un gestionnaire de bibliothèques qui permet d'ajouter rapidement les librairies nécessaires pour les capteurs, l'écran LCD et la communication réseau. Grâce au moniteur série, l'utilisateur peut visualiser les données envoyées par l'ESP32 et vérifier le bon fonctionnement du système. Cet environnement facilite ainsi le développement, le test et le débogage du programme.

Figure 9 : logo arduino IDE

1.2.2 Blynk

Blynk est une plateforme IoT destinée à la création d'applications mobiles pour le suivi et le contrôle d'objets connectés. Elle propose plusieurs composants appelés *widgets*, tels que les afficheurs de valeurs, les graphiques, les boutons et les notifications. Dans ce projet, les widgets permettent d'afficher la température, l'humidité et la qualité de l'air en temps réel. Blynk offre également un système de notification automatique qui avertit l'utilisateur lorsque les seuils de pollution sont dépassés. Son interface intuitive permet une surveillance à distance simple et efficace via un smartphone.

Figure 10 : logo Blynk

1.2.3 Firebase

Firebase est une plateforme cloud de Google utilisée pour le stockage et la gestion des données en temps réel. Elle repose principalement sur une base de données en ligne capable de synchroniser instantanément les informations entre l'ESP32 et l'application. Dans ce projet, Firebase permet d'enregistrer l'historique des mesures de qualité de l'air, de température et d'humidité. Elle inclut également des fonctions de sécurité, d'authentification et de gestion des accès, garantissant la protection des données. Firebase facilite ainsi l'analyse à long terme et l'exploitation des données collectées.

Figure 11 : logo Firebase

Conclusion

Le Chapitre II a permis de présenter l'ensemble des ressources techniques mobilisées dans le projet. Le choix des composants matériels et des logiciels a été justifié en fonction de leurs performances, leur coût et leur compatibilité avec les objectifs du système. Ces éléments forment la base matérielle et logicielle nécessaire à la réalisation pratique décrite dans le chapitre suivant.

CHAPITRE III

Réalisation du projet

1. Introduction

Ce chapitre décrit les étapes concrètes de réalisation du système, depuis le câblage des capteurs jusqu'à la configuration des plateformes cloud. Il inclut la simulation du circuit, la transmission des données vers Firebase et l'intégration avec Blynk pour la visualisation en temps réel.

1.1 Simulation de circuit

1.1.1 Câblage du DHT11

Le capteur DHT11, dédié à la mesure de température et d'humidité, est connecté directement aux ports GPIO de ESP32 avec trois connexions principales :

capteur(DHT11)	Connexion
Vcc+	Pin 5V
GND	Pin GND
Signal Pin	GPIO4

Table 1: Connexions du capteur DHT11 avec l'ESP32

1.1.2 Câblage du MQ135

Le capteur MQ135 génère un signal analogique qui doit être converti en signal numérique pour être traité par ESP32. Pour cette conversion, un convertisseur analogique-numérique MCP3008 est utilisé comme intermédiaire. Le MCP3008 est ensuite connecté aux ports GPIO de ESP32 via le protocole SPI, permettant ainsi la lecture des données de qualité d'air.

MQ135	ESP32
VCC	5 V
GND	GND
AO	GPIO 34

Table 2: Connexions du capteur MQ135 avec l'ESP32

1.1.3 Câblage du buzzer

Un buzzer est un composant électronique utilisé pour produire un signal sonore. Il permet de générer des bips ou des alertes sonores afin d'informer ou d'avertir l'utilisateur d'un événement particulier.

Buzzer	ESP32
VCC (+)	GPIO 26
GND (−)	GND

Table 3: Connexions du buzzer avec l'ESP32

1.2 Shéma éléctrique

Le schéma de câblage a été réalisé sur **Cirkit Design** afin de représenter les connexions entre l'ESP32, le capteur MQ135 et le capteur DHT11. Le MQ135 est relié à une entrée analogique de l'ESP32 pour mesurer les gaz polluants, tandis que le DHT11 est connecté à une broche numérique pour mesurer la température et l'humidité. Tous les composants partagent la même masse (GND) et sont alimentés par l'ESP32, ce qui permet de vérifier le bon fonctionnement du circuit avant sa réalisation pratique.

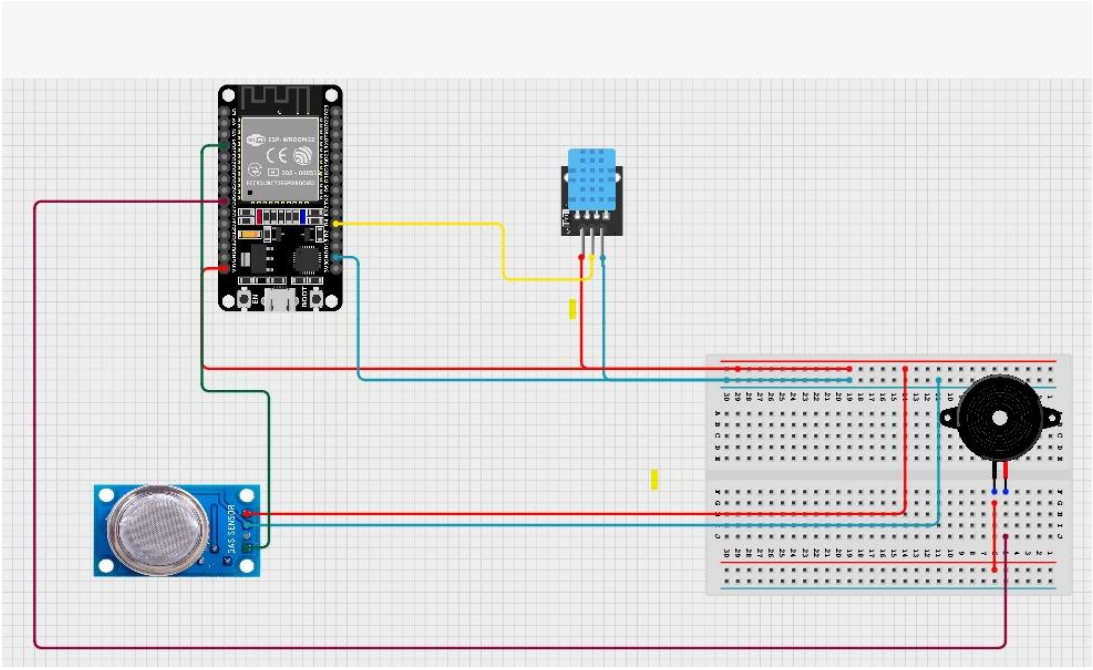

Figure 12 : shema de cablage

1.3 Circuit finale

Le circuit final intègre harmonieusement tous les composants nécessaires pour la surveillance environnementale. Sur une platine d'expérimentation, le capteur DHT11 et le capteur MQ135 sont connectés à la ESP32 , chacun avec leur configuration spécifique

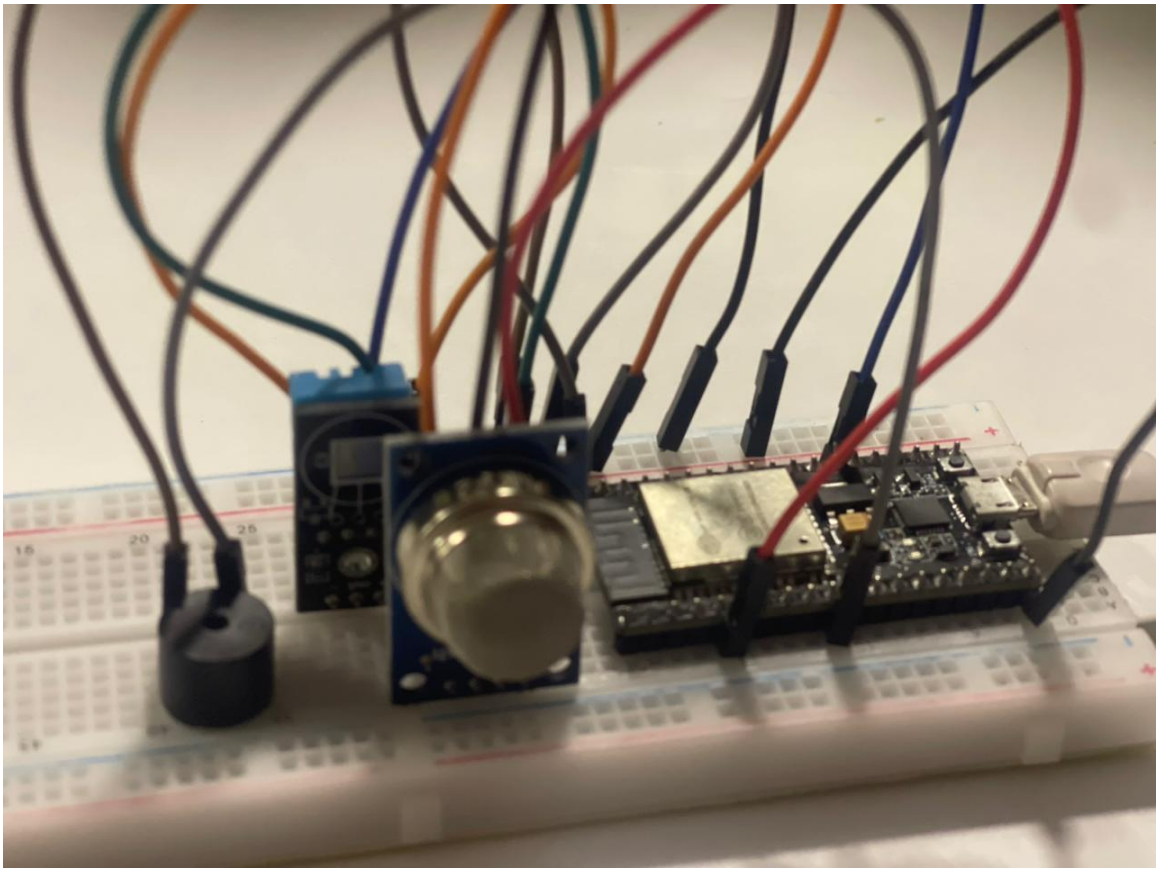

Figure 13 : shema réele

1.4 Transmission des données vers Firebase

- **Installation de la bibliothèque Firebase-Admin**

Pour permettre l'intégration avec FirebaseLa bibliothèque firebase-admin pour Python permet d'établir une connexion sécurisée avec la base de données Firebase, facilitant ainsi l'envoi et la synchronisation des données collectées par les capteurs DHT11 et MQ135 vers le cloud, grâce à des méthodes intégrées pour l'authentification et la gestion des flux de données en temps réel.

- **Création d'un projet Firebase**

Un projet a été créé sur la console Firebase pour gérer les données collectées. Cette étape a permis d'associer les données aux services Firebase tout en fournissant les outils nécessaires pour leur visualisation et gestion.

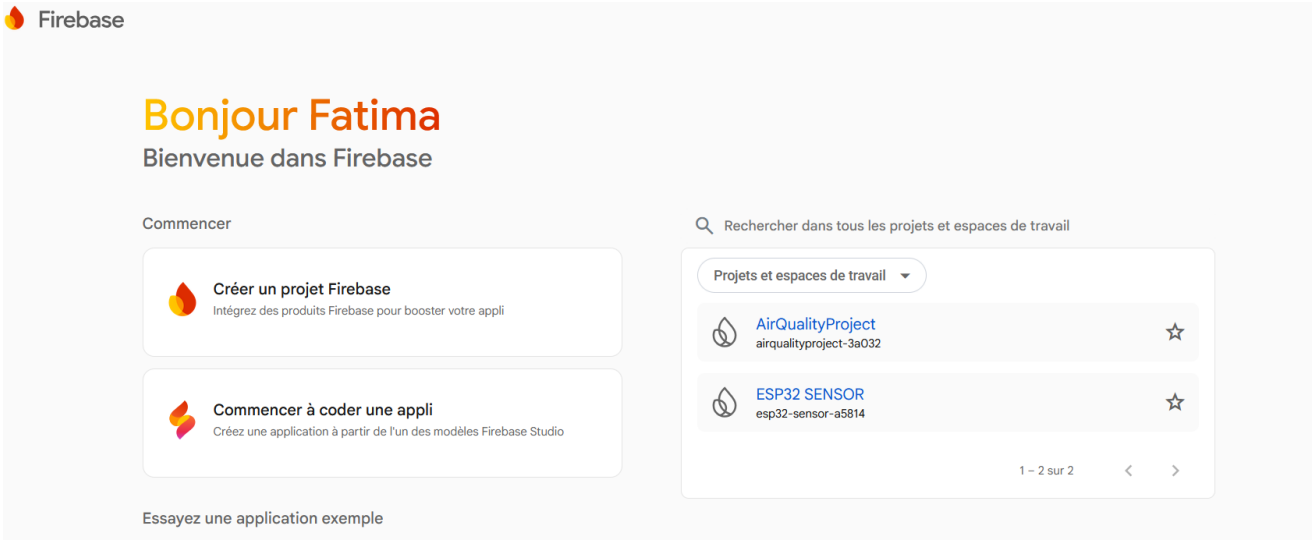

Figure 12: Création d'un projet Firebase

- **Création de la Realtime Database**

Une Realtime Database a été configurée sur Firebase pour le stockage des données collectées par le capteurs. Cette base de données, structurée sous forme de JSON, permet d'enregistrer les informations en temps réel sous forme de paires clé-valeur.

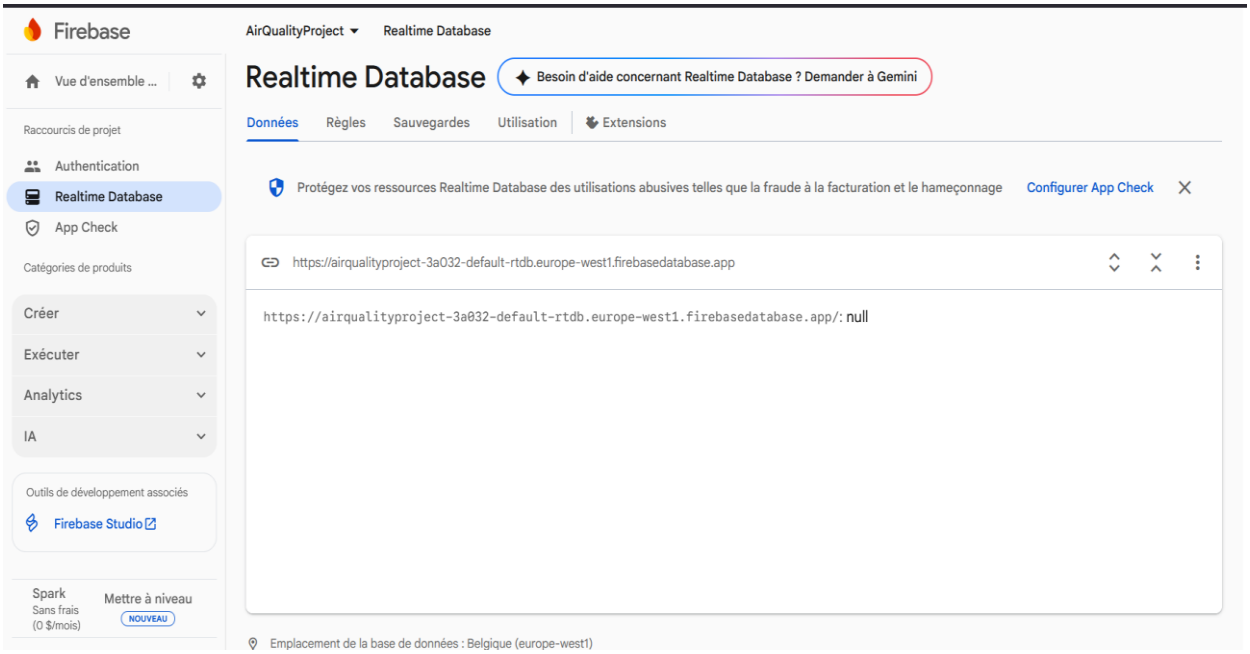

Figure 13 : Création de la Realtime Database

- **Configuration des Règles de sécurité**

Firebase utilise des règles de sécurité pour contrôler les accès en lecture et en écriture à votre base de données, définir la structure des données et gérer les index. Ces règles, hébergées sur les serveurs Firebase, sont appliquées automatiquement à chaque requête pour garantir la conformité. Par défaut, l'accès à la base de données est bloqué pour tous les utilisateurs, ce qui protège vos données contre les usages non autorisés. Ces paramètres restent en place jusqu'à ce que vous personnalisiez les règles ou configuriez un système d'authentification adapté.

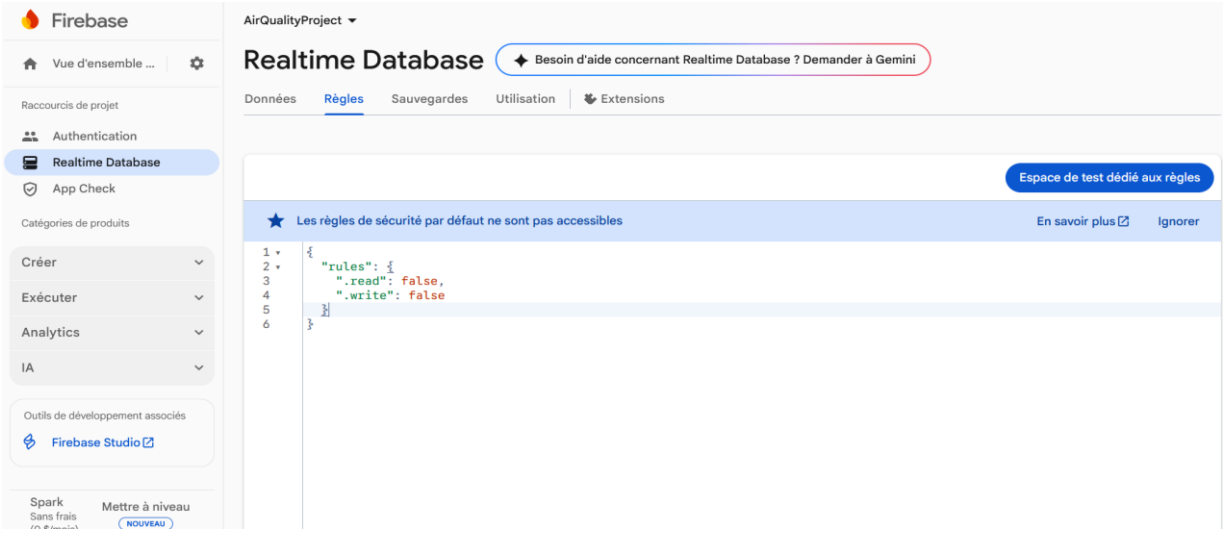

Figure 14: Configuration des Règles de sécurité

1.5 Configuration et déploiement du projet sur Blynk

Blynk est une plateforme IoT (Internet des Objets) low-code/no-code complète, permettant de connecter des dispositifs électroniques (comme des microcontrôleurs ESP32, Arduino, etc.) au cloud, de créer des applications mobiles (iOS/Android) et des tableaux de bord web pour contrôler et monitorer ces dispositifs à distance.

1.5.1 Création et vue initiale du projet dans Blynk.Console

La première étape essentielle de la réalisation du projet de surveillance de la qualité de l'air consiste à créer un Template dans la section Developer Zone de Blynk.Console. Un Template est un modèle réutilisable qui définit la configuration complète d'un type de dispositif IoT (hardware, connexion, datastreams, dashboard, etc.). Il permet de déployer facilement le projet sur un ou plusieurs dispositifs physiques.

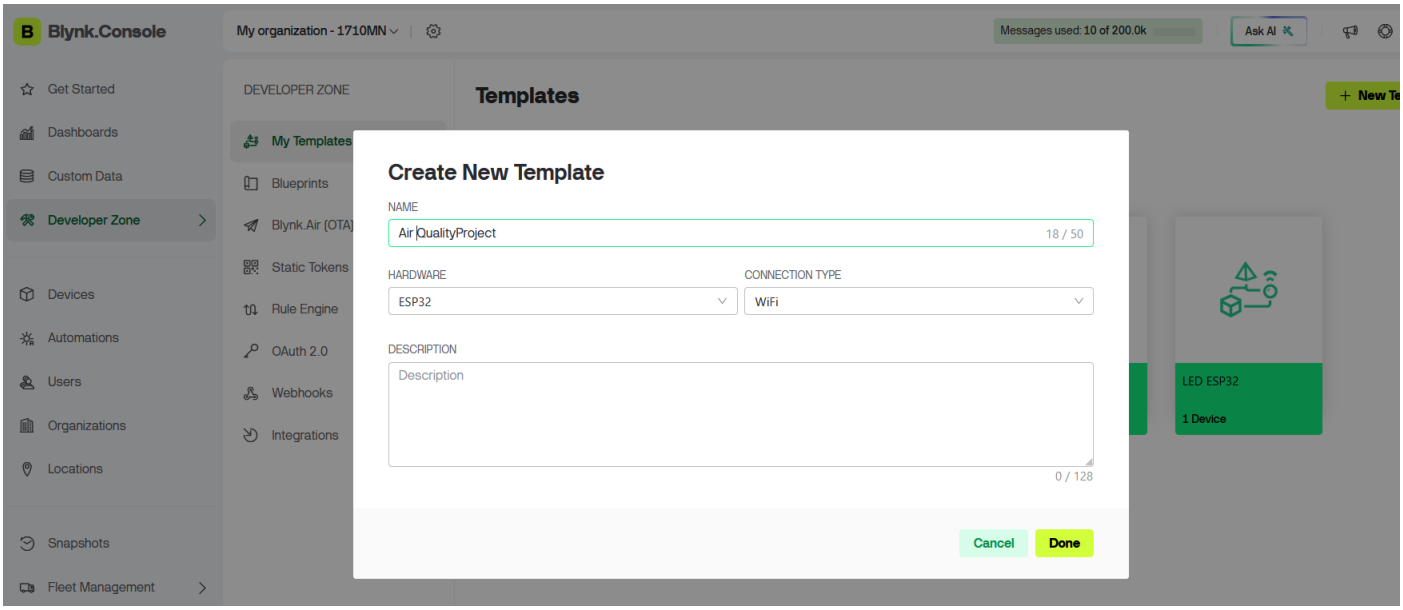

Figure 15:Création du Template "AirQualityProject" (ESP32, connexion WiFi).

1.5.2 Configuration du datastream pour la valeur de gaz

Dans cette étape, on définit le premier flux de données (datastream) via une broche virtuelle (Virtual Pin). Comme montré dans la Figure 2, un datastream nommé "GAS VALUE" est créé sur la broche virtuelle V3, de type Double (valeur décimale). La plage est fixée entre 0 (minimum) et 1000 (maximum), correspondant

typiquement aux valeurs brutes renvoyées par un capteur de gaz comme le MQ-135 ou MQ-2. Cette configuration permet à l'application Blynk de recevoir et d'afficher correctement les mesures de qualité de l'air liées aux gaz polluants. L'historique des données peut être activé si nécessaire.

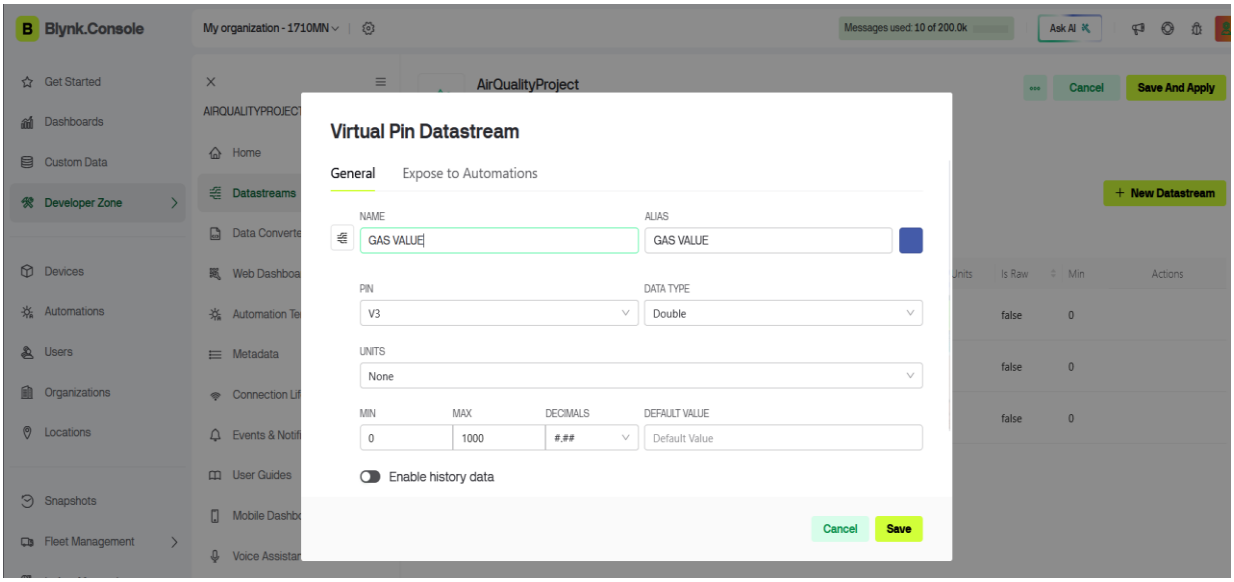

Figure 16: Configuration du datastream V3 pour la valeur de gaz (0-1000).

1.5.3 Configuration du datastream pour l'humidité

On poursuit la configuration en créant le datastream dédié à l'humidité. La Figure 3 montre la définition du datastream nommé "humidity" sur la broche virtuelle V1, également de type Double. Ici, la plage est réglée de 0 à 1 (probablement une valeur normalisée ; en pratique, il est recommandé de l'ajuster à 0-100 pour représenter le pourcentage d'humidité relatif). Ce flux de données sera utilisé pour recevoir les lectures d'un capteur d'humidité (par exemple intégré au DHT11, DHT22 ou BME280). L'alias "humidity" facilite la lisibilité dans le dashboard.

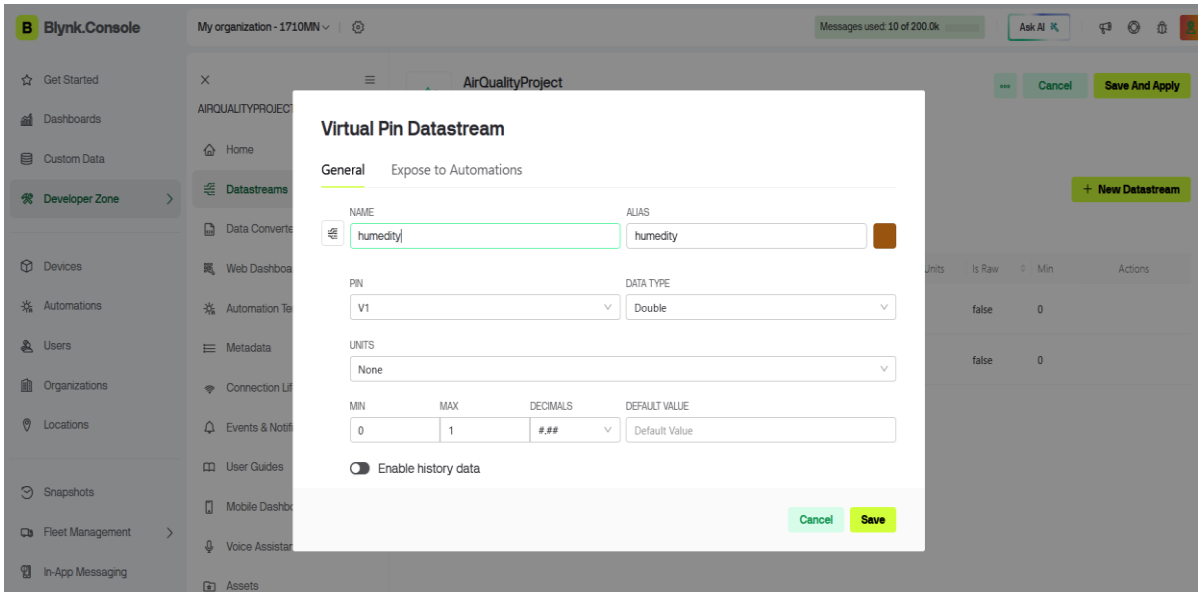

Figure 17: Configuration du datastream V1 pour l'humidité (0-1).

1.5.4 Configuration du datastream pour la température

Cette étape complète la définition des trois datastreams principaux. Dans la Figure 4, le datastream nommé "temperature" est configuré sur la broche virtuelle V2, de type Double, avec une plage de 0 à 1 (à ajuster idéalement à 0-50 ou 0-100 pour des températures en °C dans un environnement intérieur). Ce canal recevra les données de température mesurées par le capteur (DHT ou BME280). Une fois ces trois datastreams (V1, V2, V3) créés, le projet est prêt à recevoir les données envoyées par le microcontrôleur.

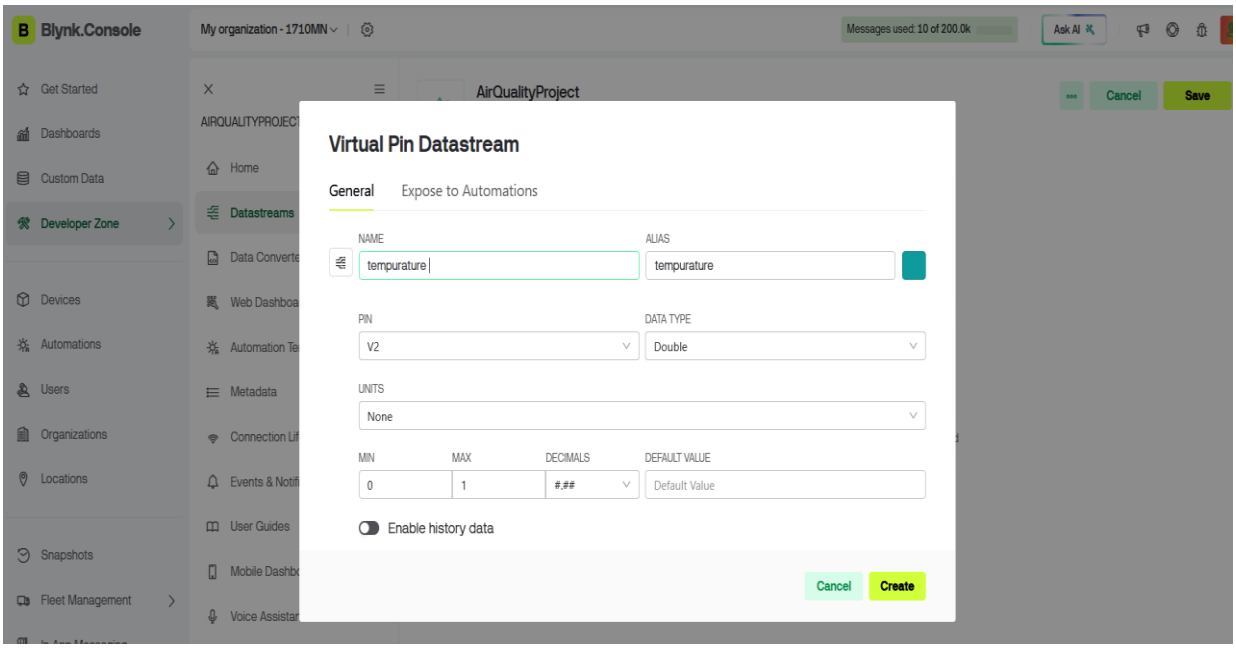

Figure 8: Configuration du datastream V2 pour la température (0-1)

1.5.5 Configuration du datastream pour l'humidité

On poursuit la configuration en créant le datastream dédié à l'humidité. La **Figure 3** montre la définition du datastream nommé "humidity" sur la broche virtuelle **V1**, également de type **Double**. Ici, la plage est réglée de **0** à **1** (probablement une valeur normalisée ; en pratique, il est recommandé de l'ajuster à 0-100 pour représenter le pourcentage d'humidité relatif). Ce flux de données sera utilisé pour recevoir les lectures d'un capteur d'humidité (par exemple intégré au DHT11, DHT22 ou BME280). L'alias "humidity" facilite la lisibilité dans le dashboard.

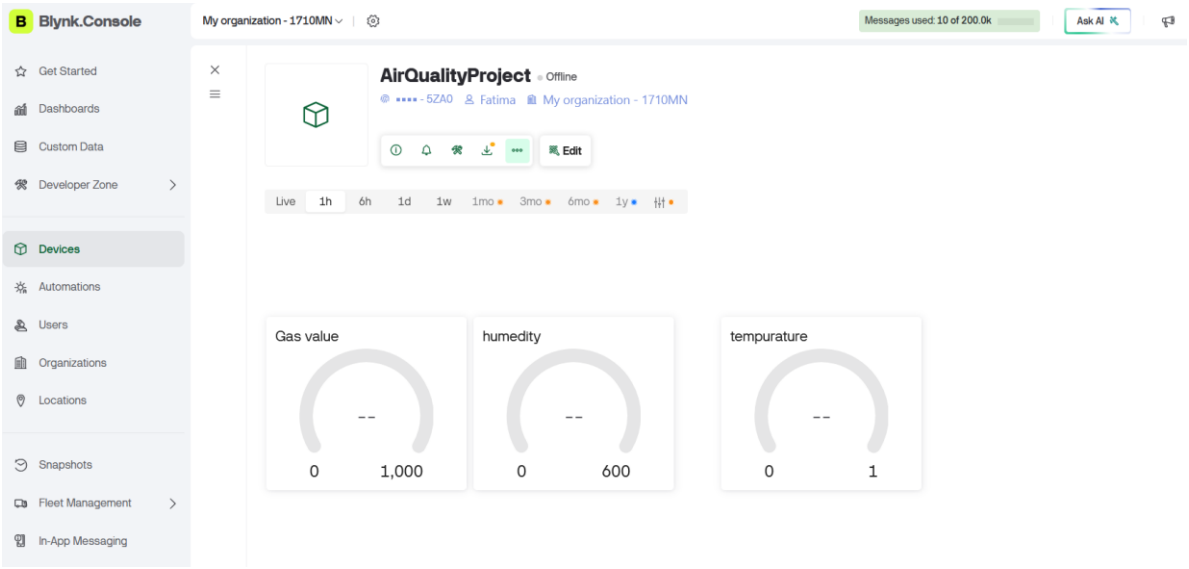

Figure 21 : Tableau de bord initial (dispositif hors ligne, valeurs "--").

1.5.6 Visualisation des données en temps réel sur le tableau de bord final

Après avoir configuré les datastreams et connecté le dispositif physique à Internet via le code Arduino/ESP32 intégrant la bibliothèque Blynk, les données commencent à arriver. La **Figure 5** présente le tableau de bord final du projet renommé "Air quality monitoring". Les jauges affichent désormais des valeurs réelles :

- **Température** : 24 (en vert, indiquant une valeur confortable),
- **Humidité** : 116 (en rouge – valeur anormale, probablement due à une mauvaise calibration de l'unité ou à une plage non ajustée en pourcentage),
- **Valeur de gaz** : 320 (en bleu, sur une échelle de 0 à 1000).

Cette étape valide le fonctionnement complet du système : les capteurs envoient les données vers les Virtual Pins correspondantes, et Blynk les affiche en temps réel sur le dashboard web et/ou l'application mobile. Des ajustements supplémentaires (calibration, ajout d'alertes ou de graphiques) peuvent être effectués par la suite.

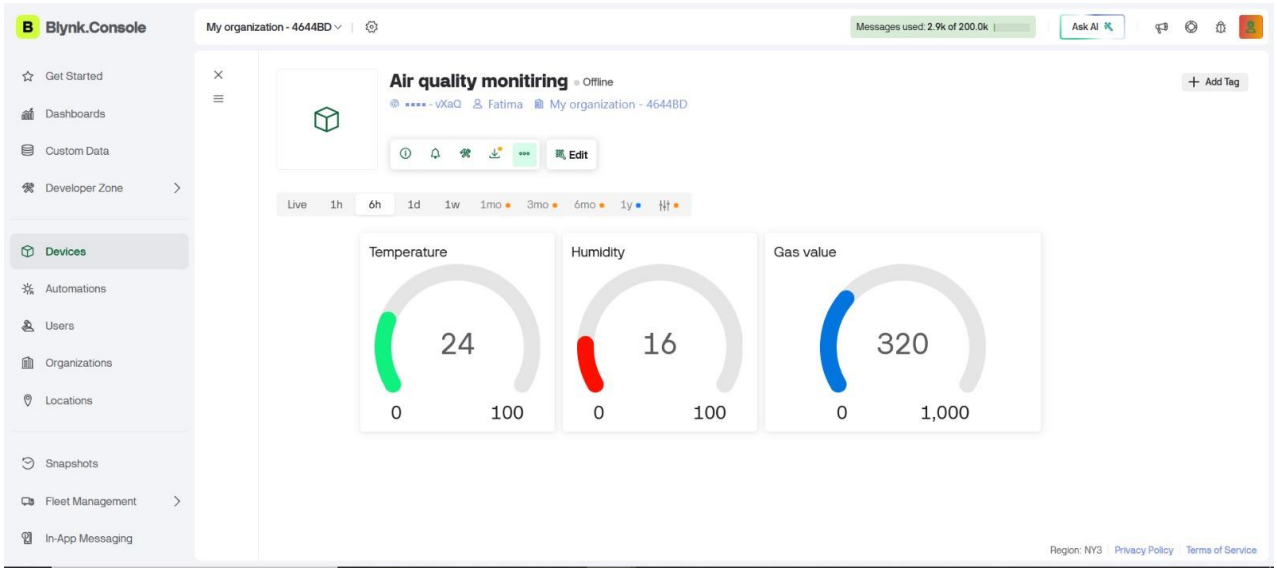

Figure 22 :Tableau de bord final avec données en temps réel (température 24, humidité 116, gaz 320).

Conclusion

Le Chapitre III a présenté la mise en œuvre complète du système de surveillance. Les étapes de câblage, de configuration des services cloud et de déploiement sur Blynk ont été suivies avec succès, permettant d'obtenir un prototype fonctionnel capable de collecter, transmettre et afficher les données environnementales en temps réel. Les résultats obtenus valident l'approche technique choisie.

Conclusion Générale

Ce projet a permis de concevoir et réaliser une sonde intelligente de surveillance de la qualité de l'air intérieur, basée sur l'IoT. Le système intègre avec succès des capteurs de gaz et d'humidité/température, un microcontrôleur ESP32, et des plateformes cloud (Firebase et Blynk) pour le stockage et la visualisation des données.

Les résultats démontrent la faisabilité d'une solution low-cost, fiable et accessible pour la surveillance environnementale en temps réel. Le dispositif offre une visibilité immédiate sur les paramètres de l'air intérieur et permet de générer des alertes en cas de dépassement des seuils recommandés.

À l'avenir, ce prototype pourrait être amélioré par l'ajout de capteurs supplémentaires (PM2.5, CO), l'intégration de l'IA pour l'analyse prédictive, ou encore le développement d'une interface web dédiée. Il constitue une base solide pour des applications dans les domaines de la santé, de l'éducation ou de la gestion énergétique des bâtiments.

En somme, ce projet illustre comment les technologies IoT peuvent être mises au service de la santé publique, en offrant des outils de prévention et de sensibilisation accessibles à tous.

BIBLIOGRAPHIE / RÉFÉRENCES

https://www.scribd.com/document/694745484/Rapport-Iot

https://youtu.be/aO92B-K4TnQ?si=zzMQWVkf4fXKDgBP

https://youtu.be/mvdpAeBVIEg?si=Z7vlzz0JBRtdHz8-

https://www.scribd.com/document/854878584/PAGE-GARDE-projet-tutore-STIC

ANNEXES

Code source du projet

1

```
#define BLYNK_PRINT Serial


// ===== BLYNK TEMPLATE INFO =====
#define BLYNK_TEMPLATE_ID "TMPL2odRNYHYF"
#define BLYNK_TEMPLATE_NAME "Air quality monitiring"
#define BLYNK_AUTH_TOKEN "lXGd3ZKEJxi9X0nGqbuSmGkZdAh2vXaQ"


#include <WiFi.h>
#include <BlynkSimpleEsp32.h>
#include "DHT.h"


// ================= WIFI =================
char ssid[] = "DESKTOP-O55IL5H 5660";
char pass[] = "123123123";


// ================= DHT =================
#define DHTPIN 4
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);


// ================= MQ135 =================
#define MQ135_PIN 34


BlynkTimer timer;
```

```
// ================= LECTURE CAPTEURS =================
void readSensors() {

  float h = dht.readHumidity();
  float t = dht.readTemperature();

  if (isnan(h) || isnan(t)) {
    Serial.println("✕ Erreur DHT11");
    return;
  }

  int air = analogRead(MQ135_PIN);

  Serial.println("----- DONNÉES -----");
  Serial.print("Température : "); Serial.println(t);
  Serial.print("Humidité    : "); Serial.println(h);
  Serial.print("Gaz (MQ135) : "); Serial.println(air);

  // Envoi vers Blynk
  Blynk.virtualWrite(V0, t);    // Température
  Blynk.virtualWrite(V1, h);    // Humidité
  Blynk.virtualWrite(V2, air);  // Gaz
}

void setup() {
  Serial.begin(9600);
```

```
  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);  // ✅ AVEC AUTH TOKEN


  dht.begin();


  timer.setInterval(2000L, readSensors); // lecture chaque 2 secondes
}


void loop() {
  Blynk.run();
  timer.run();
}
```

2

```
#include <WiFi.h>
#include <Firebase_ESP_Client.h>
#include "DHT.h"

// ================= WIFI =================
#define WIFI_SSID "win0000"
#define WIFI_PASSWORD "11111111"

// ================= FIREBASE =================
#define API_KEY "AIzaSyCtIivzNElH_uGQUj2bsmlYpxE-WG89KHc"
#define DATABASE_URL "https://esp32-sensor-a5814-default-rtdb.asia-southeast1.firebasedatabase.app/"

FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;

// ================= DHT11 =================
#define DHTPIN 4
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
```

```
// ================= MQ135 =================
#define MQ135_PIN 34

// ================= BUZZER =================
#define BUZZER_PIN 26

#define GAS_THRESHOLD 230

unsigned long lastRead = 0;

void setup() {
  Serial.begin(9600);

  pinMode(BUZZER_PIN, OUTPUT);
  digitalWrite(BUZZER_PIN, LOW);

  dht.begin();

  // ===== Connexion WiFi =====
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("Connexion WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
  }
  Serial.println("\nWiFi connecté");

  // ===== Configuration Firebase =====
  config.api_key = API_KEY;
  config.database_url = DATABASE_URL;

  Firebase.begin(&config, &auth);
  Firebase.reconnectWiFi(true);

  Serial.println("Firebase connecté");
```

```
}

void loop() {

  if (millis() - lastRead > 2000) {
    lastRead = millis();

    float h = dht.readHumidity();
    float t = dht.readTemperature();

    if (isnan(h) || isnan(t)) {
      Serial.println("✕ Erreur DHT11");
      return;
    }

    int air = analogRead(MQ135_PIN);

    Serial.println("----- DONNÉES -----");
    Serial.print("Température : "); Serial.println(t);
    Serial.print("Humidité    : "); Serial.println(h);
    Serial.print("Gaz (MQ135) : "); Serial.println(air);

    // ===== Envoi vers Firebase =====
    Firebase.RTDB.setFloat(&fbdo, "/air_monitor/temperature", t);
    Firebase.RTDB.setFloat(&fbdo, "/air_monitor/humidity", h);
    Firebase.RTDB.setInt(&fbdo, "/air_monitor/gas", air);

    // ===== Alerte Gaz =====
    if (air > GAS_THRESHOLD) {
      Serial.println(" ALERTE : CO2 ÉLEVÉ !");
      digitalWrite(BUZZER_PIN, HIGH);
      Firebase.RTDB.setString(&fbdo, "/air_monitor/alert", "CO2 ELEVE");
    } else {
      digitalWrite(BUZZER_PIN, LOW);
      Firebase.RTDB.setString(&fbdo, "/air_monitor/alert", "NORMAL");
    }
```

```
 }
}
```